```
def load_data():
  true: False
  return
    if errm in.append():
      return [i]

continuvalue:= value
```

PYTHON

EJERCICIOS

```
first_case = load_cats()
for program:
    def loWeer():
    if program = tune
    return value
```

Luis Minguillón Pascual

Python

ejercicios

Luis Minguillón Pascual

COPYRICHT

La licencia de este libro electrónico es para uso personal. Por lo tanto no puedes revenderlo a otras personas.

Gracias por respectar los derechos de autor.

©Luis Minguillón Pascual

10-06-2025

Prefacio

Este libro, *Python Ejercicios*, nace con una motivación clara: ofrecer un recurso útil y accesible para quienes desean practicar programación en Python a distintos niveles de dificultad. Desde ejercicios básicos para quienes están dando sus primeros pasos, hasta desafíos más complejos para quienes quieren afianzar y expandir sus conocimientos, el objetivo es acompañar el aprendizaje con práctica constante y variada.

El camino de la programación puede ser tan apasionante como desafiante. He vivido en carne propia las dudas del comienzo, la satisfacción de resolver un problema y el deseo de seguir aprendiendo. Por eso, decidí reunir en este libro una colección de ejercicios pensada para reforzar conceptos clave, desarrollar la lógica y fomentar la creatividad a través del código.

Quiero dedicar unas palabras especiales de agradecimiento a mi familia, por su apoyo incondicional durante la realización de este proyecto. Su paciencia, su ánimo en los momentos difíciles y su confianza en mí han sido un pilar fundamental. Sin ellos, este libro no habría sido posible.

Espero que este libro sea una herramienta valiosa para ti, lector o lectora, en tu viaje por el mundo de Python. Y sobre todo, que encuentres en la práctica no solo aprendizaje, sino también disfrute y motivación para seguir adelante.

Luis Minguillón Pascual

Indice ejercicios

1. Variables y operadores

Dificultad facil

1. Asignación de variable
2. Suma de enteros
3. Resta
4. Multiplicación
5. División
6. División entera
7. Módulo
8. Potencia
9. Operación combinada
10. Incremento de variable
11. Asignación aumentada
12. Comparación de igualdad
13. Comparación de desigualdad
14. Mayor que
15. Menor o igual
16. Operador lógico AND
17. Operador lógico OR
18. Negación lógica
19. Intercambio de variables
20. Concatenación de texto y número

Dificultad media

1. Intercambiar valores sin usar una variable temporal
2. Elevar al cubo
3. Operadores combinados
4. Calcular el área de un triángulo
5. Convertir segundos a minutos y segundos
6. Comparar edades
7. Comprobar si un número es par y mayor que 10
8. Determinar si dos números tienen el mismo resto al dividir entre 5
9. Multiplicación condicional
10. Número entre dos valores
11. Sumar dígitos de un número de dos cifras
12. Calcular promedio de 3 números
13. Resultado booleano de una expresión
14. Calcular el último dígito
15. Calcular cuántas centenas hay en un número
16. Convertir grados Celsius a Fahrenheit
17. Calcular el precio final con IVA
18. Determinar si un número tiene 3 cifras
19. Diferencia absoluta
20. Cambiar el tipo de una variable

Dificultad avanzada

1. Invertir variables sin usar una variable temporal
2. Evaluar expresiones encadenadas
3. Uso de operadores a nivel de bit
4. Operador ternario anidado
5. Multiplicación binaria de variables
6. Suma condicional en una línea
7. Cambiar de signo sin condicionales
8. Verificar si dos variables tienen el mismo valor sin usar `==`
9. Operadores de identidad
10. Modificar múltiples variables en una línea
11. Uso avanzado de `bool` y `int`
12. Evaluar prioridad de operadores
13. Operador `not` con enteros
14. Sumar los dígitos de un número usando operadores
15. Cambiar valores con XOR
16. Detectar número par sin usar `%`
17. Convertir decimal a binario sin usar `bin()`
18. Cambiar dos variables usando una lista
19. Calcular el valor absoluto sin `abs()`
20. Convertir dos números enteros en un string concatenado sin usar `str()`

2. Estructuras condicionales

Dificultad facil

1. ¿Número positivo?
2. ¿Mayor de edad?
3. Número par o impar
4. Número positivo, negativo o cero
5. Comparar dos números
6. Verificar contraseña
7. Múltiplo de 3
8. Año bisiesto (simplificado)
9. Letra vocal
10. Nota aprobada
11. Número dentro de un rango
12. Día de la semana
13. Clasificación de edad
14. Mayoría de edad en varios países
15. Número igual a 100
16. Evaluación de nota
17. ¿Es múltiplo de 2 y 3?
18. ¿Está en la lista?
19. Par o múltiplo de 5
20. ¿Palabra palíndroma?

Dificultad media

1. Determinar si un número es múltiplo de otro
2. Comparar tres números
3. Año bisiesto
4. Clasificación de edades
5. Número positivo, negativo o cero
6. Nota a letra
7. Par o impar, además múltiplo de 5
8. Contraseña segura
9. Triángulo válido
10. Número dentro de rango
11. Determinar si una letra es vocal
12. Adivinar número secreto
13. Clasificación de día
14. Número palíndromo de 3 cifras
15. Año escolar según edad
16. Cálculo de tarifa de envío
17. Evaluar examen
18. Determinar si un número es primo (versión simple)
19. Clasificación de temperatura
20. Número redondo o no

Dificultad avanzada

1. Clasificación IMC
2. Validar hora
3. Fecha válida
4. Clasificación de triángulo
5. Conversión nota a letra con validación
6. Aprobación múltiple asignaturas
7. Rango combinado
8. Verificar si tres cifras son crecientes
9. Condicional anidado: compra con descuentos
10. Número más cercano a 100
11. Año romano válido
12. Hora del día
13. Zodiaco chino
14. Edad mínima para votar
15. Aprobar por media o sobresaliente
16. Simulación de acceso a cajero
17. Verificar si es vocal y mayúscula
18. Mayor, menor y del medio
19. Clasificar carácter
20. Comparar dos fechas

3. Estructuras repetitivas, bucles

`Dificultad facil`

1. Contar del 1 al 10
2. Suma de los primeros 10 números
3. Contar hasta que se ingrese 0
4. Mostrar los múltiplos de 5 entre 1 y 50
5. Imprimir los caracteres de una palabra
6. Sumar 5 números introducidos por el usuario
7. Contar letras "a" en una frase
8. Tabla de multiplicar del 7
9. Contar cuántos números pares hay del 1 al 100
10. Imprimir números del 10 al 1
11. Sumar los números pares entre 1 y 20
12. Pedir 5 números y mostrar el mayor
13. Contar vocales en una palabra
14. Mostrar los cuadrados del 1 al 10
15. Imprimir una cuenta regresiva desde un número
16. Repetir un mensaje 5 veces
17. Leer números hasta que el usuario escriba negativo
18. Contar cuántos números positivos y negativos se ingresan
19. Sacar promedio de 4 notas
20. Mostrar los primeros 10 números impares

Dificultad media

1. Mostrar los 10 primeros múltiplos de un número dado
2. Suma de los dígitos de un número
3. Contar cuántas cifras tiene un número
4. Verificar si un número es primo
5. Generar una pirámide de asteriscos
6. Invertir una palabra
7. Verificar si una palabra es palíndroma
8. Número factorial
9. Sumar solo números pares ingresados por el usuario
10. Contar cuántos caracteres tiene una frase sin espacios
11. Imprimir los primeros n términos de la serie de Fibonacci
12. Pedir números hasta que se ingrese uno negativo, luego mostrar el promedio
13. Imprimir los divisores de un número
14. Simular 10 lanzamientos de dado y contar cuántas veces salió 6
15. Mostrar una tabla de multiplicar completa del 1 al 10
16. Detectar la primera letra duplicada en una palabra
17. Calcular el producto de los dígitos de un número
18. Contar cuántos caracteres numéricos hay en una cadena
19. Calcular cuántos números del 1 al 100 son múltiplos de 3 y 5
20. Imprimir un triángulo rectángulo invertido de n filas

Dificultad avanzada

1 Números narcisistas de 3 cifras
2. Pirámide de números centrada
3. Serie de Collatz
4. Triángulo de Pascal
5. Primos gemelos
6. Palíndromos numéricos de 5 cifras
7. Suma de dígitos hasta un solo dígito
8. Cuadrado mágico impar
9. Fracción continua
10. Serie de Fibonacci inversa
11. Factorización prima
12. Triángulo numérico creciente
13. Número más frecuente en una lista
14. Dibujar espiral cuadrada
15. Detectar ciclo en una secuencia
16. Palabras en forma de cruz
17. Dibujar tabla de multiplicar
18. Número más cercano a un promedio
19. Verificar número perfecto
20. Crear reloj digital

4. Estructuras de datos, listas

Dificultad facil

1. Crear una lista con números del 1 al 5
2. Acceder al segundo elemento de una lista
3. Cambiar un valor en una lista
4. Agregar un elemento al final de una lista
5. Insertar un elemento en una posición específica
6. Eliminar un elemento por su valor
7. Eliminar un elemento por su índice
8. Obtener la longitud de una lista
9. Ordenar una lista de números
10. Ordenar al revés
11. Verificar si un valor está en la lista
12. Sumar todos los elementos de una lista
13. Contar cuántas veces aparece un valor
14. Copiar una lista
15. Concatenar dos listas
16. Repetir una lista
17. Recorrer una lista con un bucle for
18. Obtener el índice de un elemento
19. Vaciar una lista
20. Convertir una cadena en lista

Dificultad media

1. Sumar elementos en posiciones pares
2. Eliminar duplicados
3. Encontrar el segundo mayor
4. Invertir lista sin usar reverse()
5. Separar pares e impares
6. Encontrar sublistas vacías
7. Comprobar si dos listas tienen algún elemento en común
8. Mover ceros al final
9. Aplanar una lista de listas
10. Duplicar elementos
11. Intercambiar el primer y último elemento
12. Lista sin extremos
13. Lista de cuadrados
14. Contar ocurrencias de un número
15. Insertar un elemento en el medio
16. Combinar dos listas elemento a elemento
17. Encontrar elementos únicos en una lista
18. Lista de booleanos según condición
19. Rotar lista hacia la izquierda
20. Eliminar elementos por valor

Dificultad avanzada

1. Invertir sublistas en una lista anidada
2. Eliminar duplicados preservando el orden
3. Rotar lista a la derecha `n` veces
4. Encontrar intersección sin usar conjuntos
5. Lista de acumulados
6. Eliminar todos los elementos que aparezcan más de una vez
7. Transformar lista de tuplas a lista de listas
8. Flatten (aplanar) lista anidada de profundidad 2
9. Encontrar todos los pares únicos con suma igual a `n`
10. Ordenar lista por la suma de sus sublistas
11. Crear matriz identidad de `n x n`
12. Encontrar el segundo mayor número
13. Zipear listas de distinta longitud
14. Encontrar elementos que aparecen exactamente 2 veces
15. Agrupar por índice par e impar
16. Reemplazar elementos con la media de sus vecinos
17. Contar elementos únicos sin `set`
18. Detectar si una lista es palíndromo
19. Particionar en sublistas de tamaño fijo
20. Intercalar dos listas

5. Estructura de conjuntos

Dificultad facil

1. Crear un conjunto desde una lista
2. Añadir un elemento al conjunto
3. Eliminar un elemento del conjunto
4. Comprobar si un elemento está en un conjunto
5. Unir dos conjuntos
6. Intersección de dos conjuntos
7. Diferencia de conjuntos
8. Diferencia simétrica
9. Convertir un string en conjunto
10. Longitud de un conjunto
11. Crear conjunto vacío
12. Eliminar sin error
13. Copiar un conjunto
14. Borrar todos los elementos
15. Subconjunto
16. Superconjunto
17. Iterar un conjunto
18. Sumar elementos únicos de una lista
19. Comparar dos conjuntos
20. Usar conjunto para quitar duplicados de una lista

Dificultad media

1. Crear un conjunto sin duplicados a partir de una lista
2. Verificar si un conjunto es subconjunto de otro
3. Calcular la unión de dos conjuntos
4. Calcular la intersección de dos conjuntos
5. Diferencia de conjuntos
6. Diferencia simétrica
7. Eliminar un elemento de un conjunto
8. Verificar si dos conjuntos son disjuntos
9. Congelar un conjunto
10. Iterar sobre un conjunto
11. Crear un conjunto por comprensión
12. Eliminar todos los elementos de un conjunto
13. Tamaño de un conjunto
14. Añadir elementos de una lista a un conjunto
15. Convertir cadena en conjunto de caracteres únicos
16. Comparar si dos conjuntos son iguales
17. Quitar un elemento que podría no existir
18. Crear un conjunto de pares desde una lista
19. Encontrar palabras comunes entre dos frases
20. Quitar elementos de un conjunto usando otro

Dificultad avanzada

1. Encuentra los elementos únicos comunes en tres conjuntos
2. Elimina duplicados manteniendo el orden
3. Verifica si un conjunto es subconjunto propio de otro
4. Diferencia simétrica recursiva de varios conjuntos
5. Conjunto de palabras únicas de un texto
6. Detecta conjuntos disjuntos en una lista
7. Crea un conjunto inmutable de letras
8. Intersección de todos los subconjuntos de una lista
9. Genera todas las combinaciones de pares posibles de un conjunto
10. Comprueba si dos listas contienen los mismos elementos sin importar el orden
11. Elimina todos los elementos comunes entre dos conjuntos
12. Crea el conjunto potencia de un conjunto
13. Encuentra el primer elemento duplicado en una lista
14. Reemplaza elementos de una lista si están en un conjunto
15. Comprueba si un conjunto es igual a su reverso (palíndromo lógico)
16. Intersección múltiple sin usar operadores
17. Agrupa elementos por longitud con conjuntos
18. Crea un conjunto con números no repetidos entre dos listas
19. Elimina elementos de un conjunto que cumplan condición
20. Verifica si dos conjuntos tienen la misma suma de elementos

6. Estructura de datos, tuplas

Dificultad facil

1.Crear una tupla con 3 elementos
2. Acceder al segundo elemento
3. Intentar modificar una tupla
4. Concatenar dos tuplas
5. Multiplicar una tupla
6. Comprobar si un elemento está en una tupla
7. Obtener la longitud de una tupla
8. Convertir una lista en tupla
9. Contar cuántas veces aparece un valor
10. Encontrar la posición de un elemento
11. Crear una tupla de un solo elemento
12. Desempaquetar tupla en variables
13. Usar tupla como clave de diccionario
14. Tupla dentro de una lista
15. Tupla de tuplas
16. Iterar sobre una tupla
17. Convertir tupla a string
18. Tupla vacía
19. Verificar tipo de dato
20. Sumar los valores de una tupla de enteros

Dificultad media

1. Invertir una tupla2. Encontrar todos los índices de un valor
3. Ordenar una tupla
4. Eliminar un elemento de una tupla
5. Sumar elementos por posición entre dos tuplas
6. Crear una tupla con números pares de otra tupla
7. Contar elementos únicos en una tupla
8. Agrupar elementos por pares
9. Reemplazar un valor en una tupla
10. Aplanar una tupla de tuplas
11. Convertir una cadena en tupla de caracteres
12. Intercalar dos tuplas
13. Encontrar el valor máximo sin usar `max()`
14. Crear tupla con valores únicos ordenados
15. Convertir tupla de listas en lista de tuplas
16. Encontrar pares (a, b) con suma 10
17. Contar elementos mayores que un número
18. Eliminar duplicados y mantener orden
19. Convertir lista de tuplas en diccionario
20. Combinar tupla de claves y tupla de valores

Dificultad avanzada

1. Crear una tupla de 100 números pares y contar cuántos son divisibles por 10.
2. Dada una tupla de palabras, contar cuántas tienen más de 5 letras.
3. Comprobar si una tupla contiene solo valores únicos (sin repeticiones).
4. Obtener los índices de todos los elementos 'a' en una tupla.
5. Convertir una lista de tuplas (clave, valor) en un diccionario.
6. Ordenar una tupla de tuplas por el segundo valor.
7. Intercambiar claves y valores en una tupla de pares (clave, valor).
8. Aplanar una tupla de tuplas en una sola tupla.
9. Eliminar duplicados de una tupla manteniendo el orden.
10. Sumar todas las posiciones pares de una tupla numérica.
11. Crear una tupla con los cuadrados de los primeros 30 enteros.
12. Encontrar el valor máximo y mínimo en una tupla.
13. Revertir una tupla usando slicing.
14. Comprobar si dos tuplas son anagramas (mismos elementos en diferente orden).
15. Dividir una tupla grande en sub-tuplas de 3 elementos.
16. Obtener la intersección de dos tuplas sin convertirlas en sets.
17. Encontrar los 3 elementos más frecuentes en una tupla.
18. Reemplazar un valor específico en una tupla (crear nueva).
19. Verificar si una tupla contiene solo números enteros.
20. Generar todas las permutaciones posibles de una tupla de 3 elementos.

7. Estructura de datos, diccionarios

Dificultad facil

1. Crear un diccionario simple
2. Acceder a un valor del diccionario
3. Modificar un valor
4. Añadir una nueva clave
5. Eliminar una clave
6. Ver todas las claves
7. Ver todos los valores
8. Ver todos los pares clave-valor
9. Usar un bucle `for` con `items()`
10. Comprobar si una clave existe
11. Crear un diccionario vacío
12. Diccionario de cuadrados
13. Contar letras de una palabra
14. Obtener valor con `.get()`
15. Copiar un diccionario
16. Borrar todo el diccionario
17. Fusionar dos diccionarios
18. Número de elementos del diccionario
19. Diccionario con listas
20. Iterar sobre una lista de diccionarios

Dificultad media

1. Contar la frecuencia de cada carácter en una cadena
2. Sumar los valores de un diccionario
3. Invertir un diccionario
4. Eliminar claves duplicadas por valor
5. Combinar dos diccionarios sumando valores
6. Filtrar un diccionario por valor
7. Crear un diccionario desde dos listas
8. Contar palabras en un texto
9. Ordenar diccionario por valor descendente
10. Agrupar elementos por longitud de palabra
11. Diccionario con cuadrados de números
12. Verificar si dos diccionarios son iguales
13. Contar elementos únicos en valores
14. Crear diccionario de letras y sus posicion
15. Convertir lista de tuplas a diccionario
16. Contar vocales con diccionario
17. Eliminar claves con valor nulo
18. Fusionar diccionarios con prioridad al segundo

19. Calcular promedio por estudiante
20. Crear diccionario de frecuencia de dígitos

Dificultad avanzada

1. Contar ocurrencias únicas de valores en un diccionario
2. Invertir un diccionario
3. Ordenar un diccionario por valores descendentes
4. Sumar valores de claves comunes en dos diccionarios
5. Filtrar un diccionario por valores mayores a un umbral
6. Fusionar varios diccionarios
7. Agrupar palabras por longitud
8. Detectar claves duplicadas en varios diccionarios
9. Convertir diccionario anidado en lista plana
10. Detectar valores duplicados
11. Diccionario de frecuencias de caracteres
12. Crear un índice invertido de palabras por letra inicial
13. Detectar claves con valores `None`
14. Actualizar diccionario solo si clave no existe
15. Intersección de dos diccionarios por claves
16. Diccionario a lista ordenada de tuplas
17. Diccionario de listas a lista de diccionarios
18. Reemplazar valores en un diccionario según un mapa
19. Contar número de claves únicas en varios diccionarios
20. Crear diccionario de posiciones de palabras

8. Funciones

Dificultad facil

1. Función que sume dos números
2. Función que devuelva el cuadrado de un número
3. Función que diga si un número es par
4. Función que devuelva el mayor de dos números
5. Función que concatene dos cadenas
6. Función que calcule el área de un triángulo
7. Función que cuente las letras de una cadena
8. Función que invierta una cadena
9. Función que cuente cuántas vocales hay
10. Función que calcule el factorial
11. Función que devuelva si un número es positivo, negativo o cero
12. Función que devuelva la suma de una lista
13. Función que devuelva el número más pequeño en una lista
14. Función que verifique si un carácter es vocal
15. Función que cuente cuántos elementos tiene una lista
16. Función que calcule la media de una lista
17. Función que convierta Celsius a Fahrenheit
18. Función que diga si una palabra es palíndromo

19. Función que multiplique todos los elementos de una lista
20. Función que devuelva la longitud de la palabra más larga

Dificultad media

1. Suma de una lista
2. Número primo
3. Contar vocales
4. Factorial
5. Palíndromo
6. Mayor de tres
7. Conversión Celsius a Fahrenheit
8. Número perfecto
9. Intercambiar valores
10. Contar palabras
11. Promedio de una lista
12. Números pares en una lista
13. Potencia sin usar `**`
14. Contar dígitos
15. Invertir lista
16. Buscar mayor impar
17. Números únicos
18. Producto de una lista
19. Sumar dígitos
20. Verificar anagrama

Dificultad avanzada

1. Generador de funciones derivadas numéricas
2. Memoización manual de Fibonacci
3. Aplicación de múltiples funciones
4. Generador de funciones polinómicas
5. Función inversa numérica
6. Decorador de cronómetro
7. Validación de argumentos con decorador
8. Función de orden superior para integración numérica
9. Parámetros arbitrarios de tipo `**kwargs`
10. Función curryficada
11. Aplicación recursiva sobre listas anidadas
12. Decorador de logging
13. Conversión de base numérica personalizada
14. Verificación de palíndromos recursivos
15. Conteo de palabras únicas con función
16. Simulación de funciones lambda anidadas
17. Función generadora de permutaciones
18. Compresor de cadenas simple
19. Evaluador de expresiones matemáticas simples

20. Decorador con parámetros

9. Programación orientada a objetos

Dificultad facil

1. Crear una clase `Perro`
2. Clase `Coche` con método `arrancar`
3. Clase `Rectángulo` con cálculo de área
4. Clase `Libro` con título y autor
5. Clase `CuentaBancaria` con saldo
6. Clase `Estudiante` con promedio
7. Clase `Persona` con saludo personalizado
8. Clase `Producto` con descuento
9. Clase `Circulo` con cálculo de área
10. Clase `Animal` con herencia
11. Clase `Punto` en el plano
12. Clase `Cancion` con duración
13. Clase `Usuario` con contraseña
14. Clase `Contador` que se incrementa
15. Clase `Empleado` con sueldo y bonificación
16. Clase `Reloj` que avanza el tiempo
17. Clase `Temperatura` con conversión
18. Clase `Persona` con método `es_mayor`
19. Clase `Pelicula` con método `es_larga`
20. Clase `Cuenta` con impresión de estado

Dificultad media

1. Clase Rectángulo
2. Clase Círculo con perímetro
3. Contador con incremento
4. Empleado con aumento de sueldo
5. Cuenta bancaria
6. Clase Estudiante con nota media
7. Clase Vehículo con herencia
8. Clase Persona con validación
9. Producto con descuento
10. Clase Fracción con suma
11. Inventario de productos
12. Clase Cuenta con historial
13. Clase Agenda
14. Sistema de notas por asignatura
15. Contador compartido con clase
16. Sistema de reservas
17. Clase Cronómetro
18. Simulación de dados

19. Herencia con constructor
20. Sistema de login

Dificultad avanzada

1. Sistema de Plugins Dinámicos
2. Sistema Bancario con Transacciones
3. Gestor de Tareas con Prioridades
4. Editor de Texto con Deshacer/Rehacer
5. Modelado de Árbol Genealógico
6. Sistema de Archivos con Composite Pattern
7. Sistema de Ventas con Decorador
8. Simulador de Eventos con Observador
9. Máquina de Estados Finita
10. Gestión de Recursos con Context Manager
11. Validador de Formularios
12. Sistema de Chat con Usuarios y Grupos
13. Contador de Palabras con Sobrecarga
14. Control de Robots con Polimorfismo
15. Generador de Informes con Fábrica
16. Simulador de Procesos con Clases Abstractas
17. Simulación de Red Neuronal con Nodos
18. Gestor de Roles y Permisos
19. Sistema de Publicación y Suscripción
20. Clonación de Objetos con Copia Profunda

10. Tratamiento de ficheros

Dificultad facil

1. Crear un archivo de texto y escribir una línea
2. Leer el contenido de un archivo
3. Escribir varias líneas en un archivo
4. Leer línea por línea
5. Añadir contenido a un archivo existente
6. Contar las líneas de un archivo
7. Copiar contenido de un archivo a otro
8. Comprobar si un archivo existe
9. Leer solo la primera línea
10. Leer archivo y convertir texto a mayúsculas
11. Crear un archivo solo si no existe
12. Leer archivo completo y contar caracteres
13. Eliminar un archivo
14. Escribir números del 1 al 10 en un archivo
15. Leer un archivo y convertir a lista
16. Crear un archivo CSV básico
17. Leer un archivo CSV sin librerías
18. Escribir fecha actual en un archivo

19. Crear archivo con texto invertido
20. Contar cuántas veces aparece una palabra

Dificultad media

1. Leer un archivo y contar líneas
2. Escribir una lista de cadenas en un archivo
3. Copiar contenido de un archivo a otro
4. Contar palabras en un archivo
5. Buscar y reemplazar texto en un archivo
6. Leer un archivo línea a línea sin cargar todo en memoria
7. Añadir texto al final de un archivo
8. Leer un archivo CSV y mostrar contenido
9. Escribir datos en CSV desde una lista de listas
10. Eliminar líneas vacías de un archivo
11. Crear un archivo y escribir números del 1 al 100
12. Leer archivo binario y mostrar primeros 10 bytes
13. Contar caracteres de un archivo
14. Leer y mostrar un archivo en mayúsculas
15. Extraer líneas que contienen una palabra
16. Contar cuántas veces aparece una palabra en un archivo
17. Leer archivo JSON y mostrar su contenido
18. Guardar un diccionario en archivo JSON
19. Leer archivo y contar frecuencia de palabras
20. Leer un archivo y mostrar las primeras N líneas

Dificultad avanzada

1. Leer un fichero CSV y convertirlo en una lista de diccionarios
2. Contar las líneas no vacías de un fichero de texto
3. Crear una copia cifrada de un fichero de texto (Cifrado César)
4. Unir el contenido de varios ficheros `.txt` en uno solo
5. Buscar la palabra más común en un fichero
6. Eliminar líneas duplicadas de un fichero
7. Dividir un fichero grande en partes de 100 líneas
8. Leer un JSON y acceder a una clave anidada
9. Detectar ficheros corruptos al leer archivos binarios
10. Sustituir una palabra por otra en un fichero
11. Calcular el tamaño total de todos los ficheros `.log`
12. Crear un índice de palabras con líneas donde aparecen
13. Leer un fichero Excel y mostrar los valores de una columna
14. Contar el número total de caracteres de un fichero grande sin cargarlo entero
15. Validar que un fichero CSV tiene el mismo número de columnas en cada fila
16. Analizar un log y contar errores por tipo
17. Extraer y guardar imágenes incrustadas en un fichero binario
18. Guardar los nombres únicos de un fichero en orden alfabético
19. Comprobar si dos ficheros binarios son idénticos
20. Registrar actividad de un programa con marcas de tiempo

1. Variables y operadores

Dificultad facil

1. Asignación de variable

Ejercicio:
Crea una variable llamada nombre y asígnale tu nombre. Imprime un saludo.

Solución:

```
nombre = "Ana"
print("Hola, " + nombre + "!")
```

Explicación:
Se usa = para asignar un valor. + concatena cadenas.

---

2. Suma de enteros

Ejercicio:
Declara dos variables a = 10 y b = 5. Muestra su suma.

Solución:

```
a = 10
b = 5
suma = a + b
print(suma)
```

Explicación:
El operador + suma los valores de a y b.

---

3. Resta

Ejercicio:
Calcula la diferencia entre 20 y 8.

Solución:

```
x = 20
y = 8
diferencia = x - y
print(diferencia)
```

Explicación:
- resta los valores.

---

4. Multiplicación

Ejercicio:
Multiplica 6 por 7 y muestra el resultado.

Solución:

```
a = 6
b = 7
producto = a * b
print(producto)
```

Explicación:
* multiplica los números.

---

5. División

Ejercicio:
Divide 25 entre 5.

Solución:

```
a = 25
b = 5
cociente = a / b
print(cociente)
```

Explicación:
/ da el resultado con decimales.

---

6. División entera

Ejercicio:
Obtén la división entera de 25 entre 4.

Solución:

```
a = 25
b = 4
resultado = a // b
print(resultado)
```

Explicación:
// descarta los decimales.

---

7. Módulo

Ejercicio:
Muestra el resto de dividir 17 entre 3.

Solución:

```
a = 17
b = 3
resto = a % b
print(resto)
```

Explicación:
% devuelve el resto de la división.

---

8. Potencia

Ejercicio:
Calcula 3 elevado a la 4.

Solución:

```
base = 3
exponente = 4
potencia = base ** exponente
print(potencia)
```

Explicación:
** es el operador de potencia.

---

9. Operación combinada

Ejercicio:
Calcula (2 + 3) * 4.

Solución:

```
resultado = (2 + 3) * 4
print(resultado)
```

Explicación:
Los paréntesis cambian el orden de operaciones.

---

10. Incremento de variable

Ejercicio:
Declara una variable contador = 1 y aumenta su valor en 1.

Solución:

```
contador = 1
contador = contador + 1
print(contador)
```

Explicación:
Se reasigna contador con un nuevo valor.

---

11. Asignación aumentada

Ejercicio:
Usa += para sumar 5 a la variable x = 10.

Solución:

```
x = 10
x += 5
print(x)
```

Explicación:
+= es una forma abreviada de x = x + 5.

---

12. Comparación de igualdad

Ejercicio:
Comprueba si 4 es igual a 4.

Solución:

```
print(4 == 4)
```

Explicación:
== compara si los valores son iguales.

---

13. Comparación de desigualdad

Ejercicio:
Comprueba si 4 es distinto de 5.

Solución:

```
print(4 != 5)
```

Explicación:
!= devuelve True si los valores son diferentes.

---

14. Mayor que

Ejercicio:
Comprueba si 7 es mayor que 3.

Solución:

```
print(7 > 3)
```

Explicación:
> compara si un valor es mayor que otro.

---

15. Menor o igual

Ejercicio:
Comprueba si 5 es menor o igual a 6.

Solución:

```
print(5 <= 6)
```

Explicación:
<= devuelve True si es menor o igual.

---

16. Operador lógico AND

Ejercicio:
Evalúa si 3 < 5 y 7 > 2.

Solución:

```
print(3 < 5 and 7 > 2)
```

Explicación:
and devuelve True si ambas condiciones lo son.

---

17. Operador lógico OR

Ejercicio:
Evalúa si 3 > 5 o 7 > 2.

Solución:

print(3 > 5 or 7 > 2)

Explicación:
or devuelve True si al menos una es verdadera.

---

18. Negación lógica

Ejercicio:
Niega el valor de True.

Solución:

print(not True)

Explicación:
not invierte el valor lógico.

---

19. Intercambio de variables

Ejercicio:
Intercambia los valores de a = 1 y b = 2.

Solución:

a = 1
b = 2
a, b = b, a
print(a, b)

Explicación:
Python permite intercambiar valores sin variable auxiliar.

---

20. Concatenación de texto y número

Ejercicio:
Concatena "Edad: " con el número 25.

Solución:

edad = 25
print("Edad: " + str(edad))

Explicación:
str() convierte el número en texto para poder unirlo.

Dificultad media

---

1. Intercambiar valores sin usar una variable temporal

Enunciado: Intercambia los valores de dos variables a = 5 y b = 10 sin usar una variable auxiliar.

Solución:

```
a = 5
b = 10
a, b = b, a
print(a, b)  # 10 5
```

Explicación: Python permite intercambio directo de variables usando a, b = b, a.

---

2. Elevar al cubo

Enunciado: Dada una variable x = 3, crea una nueva variable cubo que contenga su valor al cubo.

Solución:

```
x = 3
cubo = x ** 3
print(cubo)  # 27
```

Explicación: Se usa el operador de potencia **.

---

3. Operadores combinados

Enunciado: Dado x = 10, aplícale: sumar 5, restar 2, multiplicar por 3 y dividir por 2, usando operadores combinados.

Solución:

```
x = 10
x += 5
x -= 2
x *= 3
x /= 2
print(x)  # 19.5
```

Explicación: +=, -=, etc., modifican la variable en el lugar.

---

4. Calcular el área de un triángulo

Enunciado: Usa las variables base = 8 y altura = 5 para calcular el área de un triángulo.

Solución:

```
base = 8
altura = 5
area = (base * altura) / 2
print(area)  # 20.0
```

Explicación: Fórmula del área: (base × altura) / 2.

---

5. Convertir segundos a minutos y segundos

Enunciado: Dado total_segundos = 370, muestra cuántos minutos y segundos son.

Solución:

```
total_segundos = 370
minutos = total_segundos // 60
segundos = total_segundos % 60
print(minutos, "min", segundos, "seg")  # 6 min 10 seg
```

Explicación: Se usa división entera // y módulo %.

---

6. Comparar edades

Enunciado: Dadas edad1 = 24 y edad2 = 30, ¿quién es mayor?

Solución:

```
edad1 = 24
edad2 = 30
mayor = "edad1" if edad1 > edad2 else "edad2"
print(mayor)  # edad2
```

Explicación: Se usa un operador ternario condición ? valor1 : valor2.

---

7. Comprobar si un número es par y mayor que 10

Enunciado: Con n = 14, verifica si es par y mayor que 10.

Solución:

```
n = 14
es_valido = (n % 2 == 0) and (n > 10)
print(es_valido)  # True
```

Explicación: Se combinan condiciones con and.

---

8. Determinar si dos números tienen el mismo resto al dividir entre 5

Enunciado: Dados a = 17 y b = 22, determina si tienen el mismo resto al dividir por 5.

Solución:

```
a = 17
b = 22
mismo_resto = (a % 5) == (b % 5)
print(mismo_resto)  # True
```

Explicación: Ambos tienen resto 2 al dividir entre 5.

---

9. Multiplicación condicional

Enunciado: Si a es menor que b, multiplica ambos; si no, súmalos.

Solución:

```
a = 3
b = 7
resultado = a * b if a < b else a + b
print(resultado)  # 21
```

Explicación: Uso de operador ternario.

---

10. Número entre dos valores

Enunciado: Verifica si x = 15 está entre a = 10 y b = 20.

Solución:

```
x = 15
a = 10
```

```
b = 20
en_rango = a <= x <= b
print(en_rango)  # True
```

Explicación: Comparaciones encadenadas.

---

11. Sumar dígitos de un número de dos cifras

Enunciado: Dado num = 47, suma sus dígitos.

Solución:

```
num = 47
suma = (num // 10) + (num % 10)
print(suma)  # 11
```

Explicación: 47 // 10 = 4, 47 % 10 = 7.

---

12. Calcular promedio de 3 números

Enunciado: Calcula el promedio de 12, 18 y 25.

Solución:

```
a, b, c = 12, 18, 25
promedio = (a + b + c) / 3
print(promedio)  # 18.333...
```

Explicación: Suma y divide entre 3.

---

13. Resultado booleano de una expresión

Enunciado: Evalúa not (True and False) or (10 < 20)

Solución:

```
resultado = not (True and False) or (10 < 20)
print(resultado)  # True
```

Explicación: not (False) → True, 10 < 20 → True, True or True → True.

---

14. Calcular el último dígito

Enunciado: Dado num = 9876, extrae su último dígito.

Solución:

num = 9876

ultimo = num % 10

print(ultimo) # 6

Explicación: % 10 da el último dígito.

---

15. Calcular cuántas centenas hay en un número

Enunciado: En n = 4321, ¿cuántas centenas hay?

Solución:

n = 4321

centenas = (n // 100) % 10

print(centenas) # 3

Explicación: 4321 // 100 = 43, 43 % 10 = 3.

---

16. Convertir grados Celsius a Fahrenheit

Enunciado: Si c = 25, convierte a Fahrenheit.

Solución:

c = 25

f = c * 9 / 5 + 32

print(f) # 77.0

Explicación: Fórmula: °F = °C × 9/5 + 32.

---

17. Calcular el precio final con IVA

Enunciado: Si un producto cuesta 120€, añade el 21% de IVA.

Solución:

precio = 120

iva = precio * 0.21

total = precio + iva

print(total) # 145.2

Explicación: Se suma el 21% del precio.

---

18. Determinar si un número tiene 3 cifras

Enunciado: Dado n = 345, verifica si tiene tres cifras.

Solución:

```
n = 345
tres_cifras = 100 <= n <= 999
print(tres_cifras)  # True
```

Explicación: Números entre 100 y 999.

---

19. Diferencia absoluta

Enunciado: Calcula la diferencia absoluta entre a = 5 y b = 12.

Solución:

```
a = 5
b = 12
diferencia = abs(a - b)
print(diferencia)  # 7
```

Explicación: abs() da el valor absoluto.

---

20. Cambiar el tipo de una variable

Enunciado: Si n = "123", conviértelo a entero y súmale 7.

Solución:

```
n = "123"
resultado = int(n) + 7
print(resultado)  # 130
```

Explicación: Se usa int() para conversión de tipo.

Dificultad avanzado

1. Invertir variables sin usar una variable temporal

Enunciado: Intercambia el valor de dos variables a y b sin usar una tercera variable.

Solución:

```
a = 5
b = 8
a = a + b
b = a - b
a = a - b
print(a, b)  # 8 5
```

Explicación: Se usan operaciones aritméticas para guardar el valor original de a en la suma, luego se resta y se recupera.

---

2. Evaluar expresiones encadenadas

Enunciado: Evalúa y explica el resultado de a = 3 < 4 < 5 == True.

Solución:

```
a = 3 < 4 < 5 == True
print(a)  # False
```

Explicación: Python interpreta 3 < 4 < 5 == True como (3 < 4) and (4 < 5) and (5 == True), pero 5 == True es False.

---

3. Uso de operadores a nivel de bit

Enunciado: Calcula a ^ b donde a = 12 y b = 5.

Solución:

```
a = 12
b = 5
print(a ^ b)  # 9
```

Explicación: ^ realiza XOR bit a bit: 1100 ^ 0101 = 1001 (9 en decimal).

---

4. Operador ternario anidado

Enunciado: Usa un operador ternario para clasificar un número n.

Solución:

```
n = 15
resultado = "negativo" if n < 0 else "cero" if n == 0 else "positivo"
```

```
print(resultado)  # positivo
```

Explicación: Se anidan operadores ternarios para múltiples condiciones.

---

5. Multiplicación binaria de variables

Enunciado: Multiplica dos números binarios como strings y devuelve el resultado en binario.

Solución:

```
a = "1011"
b = "110"
res = bin(int(a, 2) * int(b, 2))[2:]
print(res)  # 1001110
```

Explicación: Se convierten strings a enteros binarios, se multiplican, y se devuelve a binario.

---

6. Suma condicional en una línea

Enunciado: Suma los tres números si al menos uno es par.

Solución:

```
a, b, c = 3, 5, 8
suma = a + b + c if any(x % 2 == 0 for x in (a, b, c)) else 0
print(suma)  # 16
```

Explicación: Se usa any() con expresión generadora para verificar pares.

---

7. Cambiar de signo sin condicionales

Enunciado: Cambia el signo de un número sin usar if ni -.

Solución:

```
n = 7
n = ~n + 1
print(n)  # -7
```

Explicación: ~n + 1 es la forma de obtener el negativo usando complemento a dos.

---

8. Verificar si dos variables tienen el mismo valor sin usar ==

Solución:

```
a = 10
```

b = 10

resultado = not (a ^ b)

print(resultado) # True

Explicación: Si a ^ b da 0, son iguales.

---

9. Operadores de identidad

Enunciado: ¿Qué imprime el siguiente código?

a = 256

b = 256

print(a is b) # ?

Solución:

print(a is b) # True

Explicación: Python almacena en caché los enteros pequeños.

---

10. Modificar múltiples variables en una línea

Enunciado: Intercambia 3 variables de forma rotativa: a=1, b=2, c=3 → a=3, b=1, c=2.

Solución:

a, b, c = 1, 2, 3

a, b, c = c, a, b

print(a, b, c) # 3 1 2

Explicación: Asignación múltiple simultánea.

---

11. Uso avanzado de bool y int

Enunciado: Evalúa bool(5 and 0 or 7)

Solución:

print(bool(5 and 0 or 7)) # True

Explicación: 5 and 0 → 0, 0 or 7 → 7, bool(7) → True.

---

12. Evaluar prioridad de operadores

Enunciado: Evalúa 3 + 2 * 2 ** 3

Solución:

```
print(3 + 2 * 2 ** 3)  # 19
```

Explicación: ** > * > +. Se evalúa: 2 ** 3 = 8, 2 * 8 = 16, 3 + 16 = 19.

---

13. Operador not con enteros

Enunciado: ¿Qué imprime not 0, not 1, not -1?

Solución:

```
print(not 0, not 1, not -1)  # True False False
```

Explicación: Solo 0 es considerado False.

---

14. Sumar los dígitos de un número usando operadores

Solución:

```
n = 348
suma = sum(int(d) for d in str(n))
print(suma)  # 15
```

Explicación: Se convierte el número en string y se suman sus dígitos.

---

15. Cambiar valores con XOR

Solución:

```
a = 4
b = 7
a = a ^ b
b = a ^ b
a = a ^ b
print(a, b)  # 7 4
```

Explicación: Intercambio de variables usando XOR.

---

16. Detectar número par sin usar %

Solución:

```
n = 10
es_par = (n & 1) == 0
print(es_par)  # True
```

Explicación: Un número par tiene el bit menos significativo en 0.

---

17. Convertir decimal a binario sin usar bin()

Solución:

```
n = 13
binario = ""
while n > 0:
    binario = str(n % 2) + binario
    n //= 2
print(binario)  # 1101
```

---

18. Cambiar dos variables usando una lista

Solución:

```
vars = [3, 5]
vars[0], vars[1] = vars[1], vars[0]
print(vars)  # [5, 3]
```

---

19. Calcular el valor absoluto sin abs()

Solución:

```
n = -9
n = (n ** 2) ** 0.5
print(n)  # 9.0
```

---

20. Convertir dos números enteros en un string concatenado sin usar str()

Solución:

```
a = 7
b = 4
s = "%d%d" % (a, b)
print(s)  # "74"
```

---

2. Estructuras y condicionales

Dificultad facil

1. ¿Número positivo?

Enunciado: Escribe un programa que lea un número y diga si es positivo.

Código:

```
num = int(input("Introduce un número: "))
if num > 0:
    print("Es positivo")
else:
    print("No es positivo")
```

Explicación: Se evalúa si el número es mayor que cero.

---

2. ¿Mayor de edad?

Enunciado: Pregunta la edad de una persona y di si es mayor de edad (18 años o más).

Código:

```
edad = int(input("Introduce tu edad: "))
if edad >= 18:
    print("Mayor de edad")
else:
    print("Menor de edad")
```

Explicación: Compara si la edad es 18 o más.

---

3. Número par o impar

Enunciado: Determina si un número es par o impar.

Código:

```
num = int(input("Introduce un número: "))
if num % 2 == 0:
    print("Es par")
else:
    print("Es impar")
```

Explicación: Se usa el módulo (%) para ver si el residuo al dividir entre 2 es 0.

---

4. Número positivo, negativo o cero

Código:

```
n = int(input("Introduce un número: "))
if n > 0:
    print("Positivo")
elif n < 0:
    print("Negativo")
else:
    print("Cero")
```

Explicación: Se usan if, elif, y else para cubrir las tres opciones.

---

5. Comparar dos números

Código:

```
a = int(input("Número 1: "))
b = int(input("Número 2: "))
if a > b:
    print("El mayor es", a)
elif b > a:
    print("El mayor es", b)
else:
    print("Son iguales")
```

Explicación: Compara ambos valores y maneja el caso de igualdad.

---

6. Verificar contraseña

Código:

```
contraseña = input("Introduce la contraseña: ")
if contraseña == "secreta123":
    print("Acceso permitido")
else:
```

```
    print("Contraseña incorrecta")
```

Explicación: Se compara el texto introducido con el esperado.

---

7. Múltiplo de 3

Código:

```
n = int(input("Número: "))
if n % 3 == 0:
    print("Es múltiplo de 3")
else:
    print("No es múltiplo de 3")
```

Explicación: Se usa el operador % para saber si es divisible entre 3.

---

8. Año bisiesto (simplificado)

Código:

```
año = int(input("Año: "))
if año % 4 == 0:
    print("Es bisiesto")
else:
    print("No es bisiesto")
```

Explicación: Este es un criterio básico para años bisiestos, aunque incompleto.

---

9. Letra vocal

Código:

```
letra = input("Introduce una letra: ").lower()
if letra in "aeiou":
    print("Es una vocal")
else:
    print("No es una vocal")
```

Explicación: Se convierte a minúscula y se verifica si es una vocal.

---

10. Nota aprobada

Código:

```
nota = float(input("Introduce tu nota: "))
if nota >= 5:
    print("Aprobado")
else:
    print("Suspendido")
```

Explicación: Una nota de 5 o más es suficiente para aprobar.

---

11. Número dentro de un rango

Código:

```
num = int(input("Número: "))
if 1 <= num <= 10:
    print("Está entre 1 y 10")
else:
    print("Fuera de rango")
```

Explicación: Usa una doble condición para validar el rango.

---

12. Día de la semana

Código:

```
num = int(input("Número del 1 al 7: "))
if num == 1:
    print("Lunes")
elif num == 2:
    print("Martes")
elif num == 3:
    print("Miércoles")
elif num == 4:
    print("Jueves")
elif num == 5:
    print("Viernes")
```

```
elif num == 6:
    print("Sábado")
elif num == 7:
    print("Domingo")
else:
    print("Número inválido")
```

Explicación: Se asocia un número con el día correspondiente.

---

13. Clasificación de edad

Código:

```
edad = int(input("Edad: "))
if edad < 13:
    print("Niño")
elif edad < 18:
    print("Adolescente")
else:
    print("Adulto")
```

Explicación: Se clasifican tramos de edad con condiciones encadenadas.

---

14. Mayoría de edad en varios países

Código:

```
pais = input("País (España o USA): ").lower()
edad = int(input("Edad: "))

if pais == "españa" and edad >= 18:
    print("Mayor de edad en España")
elif pais == "usa" and edad >= 21:
    print("Mayor de edad en USA")
else:
    print("Menor de edad en", pais.title())
```

Explicación: Se usa and para combinar condiciones.

---

15. Número igual a 100

Código:

```
n = int(input("Introduce un número: "))
if n == 100:
    print("¡Exacto!")
else:
    print("No es 100")
```

Explicación: Usa == para comparar igualdad exacta.

---

16. Evaluación de nota

Código:

```
nota = float(input("Nota: "))
if nota >= 9:
    print("Sobresaliente")
elif nota >= 7:
    print("Notable")
elif nota >= 5:
    print("Aprobado")
else:
    print("Suspenso")
```

Explicación: Clasifica una nota con diferentes rangos.

---

17. ¿Es múltiplo de 2 y 3?

Código:

```
n = int(input("Número: "))
if n % 2 == 0 and n % 3 == 0:
    print("Es múltiplo de 2 y 3")
else:
    print("No lo es")
```

Explicación: Se usa and para requerir ambas condiciones.

---

18. ¿Está en la lista?

Código:

```
fruta = input("Fruta: ").lower()
if fruta in ["manzana", "pera", "plátano"]:
    print("Está en la lista")
else:
    print("No está en la lista")
```

Explicación: Se usa in para comprobar pertenencia a una lista.

---

19. Par o múltiplo de 5

Código:

```
n = int(input("Número: "))
if n % 2 == 0 or n % 5 == 0:
    print("Es par o múltiplo de 5")
else:
    print("No cumple ninguna")
```

Explicación: Usa or para que se cumpla una u otra condición.

---

20. ¿Palabra palíndroma?

Código:

```
palabra = input("Palabra: ").lower()
if palabra == palabra[::-1]:
    print("Es un palíndromo")
else:
    print("No lo es")
```

Explicación: Compara la palabra con su reverso para saber si se lee igual al derecho y al revés.

Dificultad media

---

1. Determinar si un número es múltiplo de otro

Enunciado: Escribe un programa que pida dos números y diga si el primero es múltiplo del segundo.

Explicación: Se usa if con el operador %.

Solución:

```
a = int(input("Introduce el primer número: "))
b = int(input("Introduce el segundo número: "))

if b != 0 and a % b == 0:
    print(f"{a} es múltiplo de {b}")
else:
    print(f"{a} no es múltiplo de {b}")
```

---

2. Comparar tres números

Enunciado: Escribe un programa que pida tres números y diga cuál es el mayor.

Solución:

```
a = int(input("Primer número: "))
b = int(input("Segundo número: "))
c = int(input("Tercer número: "))

if a >= b and a >= c:
    mayor = a
elif b >= a and b >= c:
    mayor = b
else:
    mayor = c

print("El mayor es:", mayor)
```

---

3. Año bisiesto

Enunciado: Escribe un programa que determine si un año es bisiesto.

Regla: Es bisiesto si es divisible por 4, pero no por 100, salvo que también sea divisible por 400.

Solución:

```
año = int(input("Introduce un año: "))

if (año % 4 == 0 and año % 100 != 0) or (año % 400 == 0):
    print("Es un año bisiesto.")
else:
    print("No es un año bisiesto.")
```

---

4. Clasificación de edades

Enunciado: Escribe un programa que clasifique la edad ingresada:

- Menor de 13: niño
- 13 a 17: adolescente
- 18 a 64: adulto
- 65 o más: anciano

Solución:

```
edad = int(input("Introduce la edad: "))

if edad < 13:
    print("Niño")
elif edad < 18:
    print("Adolescente")
elif edad < 65:
    print("Adulto")
else:
    print("Anciano")
```

---

5. Número positivo, negativo o cero

Solución:

```
num = int(input("Introduce un número: "))
```

```
if num > 0:
    print("Positivo")
elif num < 0:
    print("Negativo")
else:
    print("Cero")
```

---

6. Nota a letra

Enunciado: Escribe un programa que transforme una nota numérica entre 0 y 10 en una letra:

- 9-10: A
- 7-8.9: B
- 6-6.9: C
- 5-5.9: D
- Menos de 5: F

Solución:

```
nota = float(input("Introduce tu nota (0-10): "))

if nota >= 9:
    print("A")
elif nota >= 7:
    print("B")
elif nota >= 6:
    print("C")
elif nota >= 5:
    print("D")
else:
    print("F")
```

---

7. Par o impar, además múltiplo de 5

Solución:

```
n = int(input("Introduce un número: "))
```

```
if n % 2 == 0:
    if n % 5 == 0:
        print("Par y múltiplo de 5")
    else:
        print("Par")
else:
    if n % 5 == 0:
        print("Impar y múltiplo de 5")
    else:
        print("Impar")
```

---

8. Contraseña segura

Enunciado: Verifica si una contraseña tiene al menos 8 caracteres y contiene un número.

Solución:

```
clave = input("Introduce tu contraseña: ")

if len(clave) >= 8 and any(c.isdigit() for c in clave):
    print("Contraseña segura")
else:
    print("Contraseña insegura")
```

---

9. Triángulo válido

Enunciado: Verifica si tres lados pueden formar un triángulo.

Solución:

```
a = int(input("Lado A: "))
b = int(input("Lado B: "))
c = int(input("Lado C: "))

if a + b > c and a + c > b and b + c > a:
    print("Es un triángulo válido.")
```

```
else:
    print("No es un triángulo.")
```

---

10. Número dentro de rango

Enunciado: Verifica si un número está entre 100 y 200 inclusive.

Solución:

```
n = int(input("Introduce un número: "))

if 100 <= n <= 200:
    print("Está en el rango.")
else:
    print("Fuera de rango.")
```

---

11. Determinar si una letra es vocal

Solución:

```
letra = input("Introduce una letra: ").lower()

if letra in 'aeiou':
    print("Es una vocal.")
else:
    print("No es vocal.")
```

---

12. Adivinar número secreto

```
secreto = 7
n = int(input("Adivina el número (1-10): "))

if n == secreto:
    print("¡Correcto!")
elif n < secreto:
    print("Demasiado bajo.")
else:
```

```
    print("Demasiado alto.")
```

---

13. Clasificación de día

```
dia = input("Introduce un día de la semana: ").lower()

if dia in ['lunes', 'martes', 'miércoles', 'jueves', 'viernes']:
    print("Es día laborable.")
elif dia in ['sábado', 'domingo']:
    print("Es fin de semana.")
else:
    print("Día no válido.")
```

---

14. Número palíndromo de 3 cifras

```
n = input("Introduce un número de 3 cifras: ")

if n == n[::-1]:
    print("Es palíndromo.")
else:
    print("No lo es.")
```

---

15. Año escolar según edad

```
edad = int(input("Edad del niño: "))

if edad < 3:
    print("Demasiado joven")
elif edad <= 5:
    print("Educación Infantil")
elif edad <= 11:
    print("Primaria")
elif edad <= 16:
    print("Secundaria")
```

```
else:
    print("Fuera del sistema obligatorio")
```

---

16. Cálculo de tarifa de envío

```
peso = float(input("Peso del paquete en kg: "))

if peso <= 1:
    tarifa = 5
elif peso <= 5:
    tarifa = 10
else:
    tarifa = 20

print("Tarifa:", tarifa, "euros")
```

---

17. Evaluar examen

```
aciertos = int(input("¿Cuántas respuestas correctas? (0-20): "))

if aciertos >= 18:
    print("Excelente")
elif aciertos >= 15:
    print("Muy bien")
elif aciertos >= 10:
    print("Aprobado")
else:
    print("Suspendido")
```

---

18. Determinar si un número es primo (versión simple)

```
n = int(input("Introduce un número mayor que 1: "))

es_primo = True
```

```
for i in range(2, int(n**0.5)+1):
    if n % i == 0:
        es_primo = False
        break

if es_primo:
    print("Es primo")
else:
    print("No es primo")
```

---

19. Clasificación de temperatura

```
t = float(input("Temperatura en ºC: "))

if t < 0:
    print("Muy frío")
elif t < 15:
    print("Frío")
elif t < 25:
    print("Templado")
else:
    print("Caluroso")
```

---

20. Número redondo o no

```
num = int(input("Introduce un número: "))

if num % 10 == 0:
    print("Número redondo")
else:
    print("No es redondo")
```

Dificultad avanzada

---

1. ***Clasificación IMC***

Enunciado: *Pide peso y estatura, calcula el IMC y clasifica según:*

- *IMC < 18.5: Bajo peso*
- *18.5 ≤ IMC < 25: Normal*
- *25 ≤ IMC < 30: Sobrepeso*
- *IMC ≥ 30: Obesidad*

Solución:

```
peso = float(input("Peso (kg): "))
altura = float(input("Estatura (m): "))

if altura > 0:
    imc = peso / (altura ** 2)

    if imc < 18.5:
        print("Bajo peso")
    elif imc < 25:
        print("Peso normal")
    elif imc < 30:
        print("Sobrepeso")
    else:
        print("Obesidad")
else:
    print("Altura inválida")
```

---

2. ***Validar hora***

Enunciado: *Verifica si una hora (HH:MM) es válida.*

Solución:

```
hora = input("Introduce la hora (HH:MM): ")

try:
    h, m = map(int, hora.split(":"))
    if 0 <= h < 24 and 0 <= m < 60:
        print("Hora válida")
    else:
        print("Hora inválida")
except:
    print("Formato incorrecto")
```

---

3. ***Fecha válida***

Enunciado: *Pide día, mes y año. Verifica si la fecha es válida (sin usar datetime).*

Solución:

```
dia = int(input("Día: "))
mes = int(input("Mes: "))
año = int(input("Año: "))

bisiesto = (año % 4 == 0 and año % 100 != 0) or (año % 400 == 0)
dias_mes = [31, 29 if bisiesto else 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]

if 1 <= mes <= 12 and 1 <= dia <= dias_mes[mes - 1]:
    print("Fecha válida")
else:
    print("Fecha inválida")
```

---

4. ***Clasificación de triángulo***

Enunciado: *Pide 3 lados y clasifica el triángulo como equilátero, isósceles o escaleno, si es válido.*

Solución:

```
a = float(input("Lado A: "))
b = float(input("Lado B: "))
c = float(input("Lado C: "))

if a + b > c and a + c > b and b + c > a:
    if a == b == c:
        print("Equilátero")
    elif a == b or b == c or a == c:
        print("Isósceles")
    else:
        print("Escaleno")
else:
    print("No es un triángulo válido")
```

---

5. ***Conversión nota a letra con validación***

Solución:

```
nota = input("Introduce la nota (0-100): ")

if nota.isdigit():
    nota = int(nota)
```

```
    if 0 <= nota <= 100:
        if nota >= 90:
            print("A")
        elif nota >= 80:
            print("B")
        elif nota >= 70:
            print("C")
        elif nota >= 60:
            print("D")
        else:
            print("F")
    else:
        print("Nota fuera de rango")
else:
    print("Entrada no válida")
```

---

6. ***Aprobación múltiple asignaturas***

Enunciado: *Pide nota de tres asignaturas. Si aprueba todas (≥5), imprime "Aprobado general", si falla una: "Recuperación", si falla más de una: "Repetidor".*

Solución:

```
n1 = float(input("Nota 1: "))
n2 = float(input("Nota 2: "))
n3 = float(input("Nota 3: "))

suspensos = 0
for n in [n1, n2, n3]:
    if n < 5:
        suspensos += 1

if suspensos == 0:
    print("Aprobado general")
elif suspensos == 1:
    print("Recuperación")
else:
    print("Repetidor")
```

---

7. ***Rango combinado***

Enunciado: *Dado un número, determinar si está fuera de todos estos rangos: (10-20), (30-40) o (50-60)*

Solución:

```
x = int(input("Introduce un número: "))
```

```
if not (10 < x < 20 or 30 < x < 40 or 50 < x < 60):
    print("Está fuera de todos los rangos")
else:
    print("Está dentro de algún rango")
```

---

8. ***Verificar si tres cifras son crecientes***

```
num = input("Introduce un número de 3 cifras: ")

if len(num) == 3 and num.isdigit():
    if num[0] < num[1] < num[2]:
        print("Cifras crecientes")
    else:
        print("No son crecientes")
else:
    print("Entrada inválida")
```

---

9. ***Condicional anidado: compra con descuentos***

```
compra = float(input("Total de la compra (€): "))
cliente = input("¿Es cliente premium? (s/n): ").lower()

if compra >= 100:
    if cliente == 's':
        descuento = 0.25
    else:
        descuento = 0.10
elif compra >= 50:
    descuento = 0.05
else:
    descuento = 0

final = compra * (1 - descuento)
print(f"Total con descuento: {final:.2f} €")
```

---

10. ***Número más cercano a 100***

```
a = int(input("Número A: "))
b = int(input("Número B: "))

if abs(100 - a) < abs(100 - b):
    print("A está más cerca de 100")
elif abs(100 - a) > abs(100 - b):
```

```
    print("B está más cerca de 100")
else:
    print("Ambos están a la misma distancia")
```

---

11. ***Año romano válido***

```
año = input("Introduce un año en números romanos (ej. MCMXC): ").upper()

valores = {'I':1, 'V':5, 'X':10, 'L':50, 'C':100, 'D':500, 'M':1000}

valido = all(c in valores for c in año)
print("Válido" if valido else "Inválido")
```

---

12. ***Hora del día***

```
h = int(input("Hora (0-23): "))

if 6 <= h < 12:
    print("Buenos días")
elif 12 <= h < 18:
    print("Buenas tardes")
elif 18 <= h < 22:
    print("Buenas noches")
else:
    print("Es madrugada")
```

---

13. ***Zodiaco chino***

```
año = int(input("Introduce tu año de nacimiento: "))
signos = ["Mono", "Gallo", "Perro", "Cerdo", "Rata", "Buey",
          "Tigre", "Conejo", "Dragón", "Serpiente", "Caballo", "Cabra"]
print("Tu signo es:", signos[año % 12])
```

---

14. ***Edad mínima para votar***

```
edad = int(input("Edad: "))
pais = input("País (es/mx/us): ").lower()

if pais == 'es' and edad >= 18:
    print("Puede votar")
elif pais == 'mx' and edad >= 18:
    print("Puede votar")
```

```
elif pais == 'us' and edad >= 18:
    print("Puede votar")
else:
    print("No puede votar")
```

---

15. ***Aprobar por media o sobresaliente***

```
n1 = float(input("Nota 1: "))
n2 = float(input("Nota 2: "))

media = (n1 + n2) / 2

if media >= 5 or n1 >= 9 or n2 >= 9:
    print("Aprobado")
else:
    print("Suspendido")
```

---

16. ***Simulación de acceso a cajero***

```
clave = "1234"
entrada = input("Introduce la clave: ")

if entrada == clave:
    print("Acceso concedido")
else:
    print("Acceso denegado")
```

---

17. ***Verificar si es vocal y mayúscula***

```
letra = input("Introduce una letra: ")

if letra in "AEIOU":
    print("Vocal mayúscula")
elif letra in "aeiou":
    print("Vocal minúscula")
else:
    print("No es vocal")
```

---

18. ***Mayor, menor y del medio***

```
a = int(input("A: "))
b = int(input("B: "))
```

```
c = int(input("C: "))

mayor = max(a, b, c)
menor = min(a, b, c)
medio = a + b + c - mayor - menor

print(f"Mayor: {mayor}, Medio: {medio}, Menor: {menor}")
```

---

19. ***Clasificar carácter***

```
car = input("Introduce un carácter: ")

if car.isdigit():
    print("Es un número")
elif car.isalpha():
    print("Es una letra")
else:
    print("Otro símbolo")
```

---

20. ***Comparar dos fechas***

```
d1 = int(input("Día 1: "))
m1 = int(input("Mes 1: "))
a1 = int(input("Año 1: "))

d2 = int(input("Día 2: "))
m2 = int(input("Mes 2: "))
a2 = int(input("Año 2: "))

if (a1, m1, d1) < (a2, m2, d2):
    print("Fecha 1 es anterior")
elif (a1, m1, d1) > (a2, m2, d2):
    print("Fecha 2 es anterior")
else:
    print("Fechas iguales")
```

---

3. Estructuras repetitivas, bucles

Dificultad faci*l*

---

1. Contar del 1 al 10

Enunciado: Imprime los números del 1 al 10 usando un bucle for.

Solución:

```
for i in range(1, 11):
    print(i)
```

Explicación: range(1, 11) genera los números del 1 al 10.

---

2. Suma de los primeros 10 números

Enunciado: Suma los números del 1 al 10 e imprime el resultado.

Solución:

```
suma = 0
for i in range(1, 11):
    suma += i
print("Suma:", suma)
```

Explicación: Se acumulan los valores del 1 al 10 en la variable suma.

---

3. Contar hasta que se ingrese 0

Enunciado: Pide números al usuario hasta que escriba 0. Luego, imprime cuántos números introdujo.

Solución:

```
contador = 0
while True:
    numero = int(input("Ingresa un número (0 para salir): "))
    if numero == 0:
        break
    contador += 1
print("Cantidad de números ingresados:", contador)
```

Explicación: Se usa un while con break para salir cuando el usuario escribe 0.

---

4. Mostrar los múltiplos de 5 entre 1 y 50

Solución:

```
for i in range(1, 51):
    if i % 5 == 0:
        print(i)
```

Explicación: Se imprimen solo los números divisibles entre 5.

---

5. Imprimir los caracteres de una palabra

Enunciado: Pide una palabra e imprime cada letra por separado.

Solución:

```
palabra = input("Introduce una palabra: ")
for letra in palabra:
    print(letra)
```

Explicación: Se recorre la cadena carácter por carácter.

---

6. Sumar 5 números introducidos por el usuario

Solución:

```
suma = 0
for i in range(5):
    numero = int(input("Introduce un número: "))
    suma += numero
print("La suma es:", suma)
```

Explicación: Se pide 5 veces un número y se suman.

---

7. Contar letras "a" en una frase

Solución:

```
frase = input("Introduce una frase: ")
contador = 0
for letra in frase:
    if letra == "a":
        contador += 1
print("Cantidad de letras 'a':", contador)
```

Explicación: Se cuenta cuántas veces aparece la letra "a".

---

8. Tabla de multiplicar del 7

Solución:

```
for i in range(1, 11):
    print(f"7 x {i} = {7 * i}")
```

Explicación: Se muestra la tabla del 7 del 1 al 10.

---

9. Contar cuántos números pares hay del 1 al 100

Solución:

```
contador = 0
for i in range(1, 101):
    if i % 2 == 0:
        contador += 1
print("Cantidad de pares:", contador)
```

Explicación: Se verifica si cada número es divisible entre 2.

---

10. Imprimir números del 10 al 1

Solución:

```
for i in range(10, 0, -1):
    print(i)
```

Explicación: range(10, 0, -1) cuenta hacia atrás.

---

11. Sumar los números pares entre 1 y 20

Solución:

```
suma = 0
for i in range(2, 21, 2):
    suma += i
print("Suma de pares:", suma)
```

Explicación: Se recorre solo números pares con salto de 2.

---

12. Pedir 5 números y mostrar el mayor

Solución:

```
mayor = None
for _ in range(5):
    n = int(input("Número: "))
    if mayor is None or n > mayor:
        mayor = n
print("El mayor es:", mayor)
```

Explicación: Se compara cada número con el actual mayor.

---

13. Contar vocales en una palabra

Solución:

```
palabra = input("Introduce una palabra: ")
vocales = "aeiou"
contador = 0
for letra in palabra.lower():
    if letra in vocales:
        contador += 1
print("Cantidad de vocales:", contador)
```

Explicación: Se cuentan letras si están en el conjunto de vocales.

---

14. Mostrar los cuadrados del 1 al 10

Solución:

```
for i in range(1, 11):
    print(f"{i}^2 = {i**2}")
```

Explicación: Se calcula el cuadrado con i**2.

---

15. Imprimir una cuenta regresiva desde un número

Solución:

```
n = int(input("Desde qué número quieres contar atrás? "))
for i in range(n, -1, -1):
    print(i)
```

Explicación: Se inicia en n y va hasta 0.

---

16. Repetir un mensaje 5 veces

Solución:

```
for _ in range(5):
    print("¡Hola mundo!")
```

Explicación: _ se usa cuando no se necesita el índice.

---

17. Leer números hasta que el usuario escriba negativo

Solución:

```
while True:
    numero = int(input("Introduce un número (negativo para salir): "))
    if numero < 0:
        break
```

Explicación: Se usa break para salir del bucle al ingresar un número negativo.

---

18. Contar cuántos números positivos y negativos se ingresan

Solución:

```
positivos = 0
negativos = 0
for _ in range(5):
    n = int(input("Número: "))
    if n > 0:
        positivos += 1
    elif n < 0:
        negativos += 1
print("Positivos:", positivos)
print("Negativos:", negativos)
```

Explicación: Se verifica el signo de cada número ingresado.

---

19. Sacar promedio de 4 notas

Solución:

```
suma = 0
for _ in range(4):
    nota = float(input("Introduce una nota: "))
    suma += nota
promedio = suma / 4
print("Promedio:", promedio)
```

Explicación: Se suman las notas y se dividen por 4.

---

20. Mostrar los primeros 10 números impares

Solución:

```
for i in range(1, 20, 2):
    print(i)
```

Explicación: Se generan impares con un range que salta de 2 en 2 desde 1.

Dificultad media

1. Mostrar los 10 primeros múltiplos de un número dado

```
num = int(input("Introduce un número: "))
for i in range(1, 11):
    print(f"{num} x {i} = {num * i}")
```

Explicación: Se multiplica el número por los 10 primeros enteros positivos.

---

2. Suma de los dígitos de un número

```
numero = int(input("Número: "))
suma = 0
while numero > 0:
    suma += numero % 10
    numero //= 10
print("Suma de dígitos:", suma)
```

Explicación: Extrae cada dígito con % y reduce el número con //.

---

3. Contar cuántas cifras tiene un número

```
numero = int(input("Número: "))
contador = 0
while numero > 0:
    numero //= 10
    contador += 1
print("Cantidad de cifras:", contador)
```

Explicación: Se divide entre 10 hasta que no queden dígitos.

---

4. Verificar si un número es primo

```
n = int(input("Número: "))
es_primo = True
if n < 2:
    es_primo = False
```

```
else:
    for i in range(2, int(n**0.5) + 1):
        if n % i == 0:
            es_primo = False
            break
print("Es primo:", es_primo)
```

Explicación: Un número no es primo si tiene divisores entre 2 y √n.

---

5. Generar una pirámide de asteriscos

```
filas = int(input("Número de filas: "))
for i in range(1, filas + 1):
    print("*" * i)
```

Explicación: Se imprime una línea con i asteriscos por fila.

---

6. Invertir una palabra

```
palabra = input("Palabra: ")
invertida = ""
for letra in palabra:
    invertida = letra + invertida
print("Invertida:", invertida)
```

Explicación: Se antepone cada letra a la variable invertida.

---

7. Verificar si una palabra es palíndroma

```
palabra = input("Palabra: ").lower()
es_palindroma = palabra == palabra[::-1]
print("¿Es palíndroma?", es_palindroma)
```

Explicación: Compara la palabra con su versión invertida.

---

8. Número factorial

```
n = int(input("Número: "))
fact = 1
for i in range(1, n + 1):
    fact *= i
print("Factorial:", fact)
```

Explicación: Se multiplica del 1 a n.

---

9. Sumar solo números pares ingresados por el usuario

```
suma = 0
for _ in range(5):
    n = int(input("Número: "))
    if n % 2 == 0:
        suma += n
print("Suma de pares:", suma)
```

Explicación: Solo se suman si el número es divisible entre 2.

---

10. Contar cuántos caracteres tiene una frase sin espacios

```
frase = input("Frase: ")
contador = 0
for c in frase:
    if c != ' ':
        contador += 1
print("Caracteres sin espacios:", contador)
```

Explicación: Ignora los espacios al contar.

---

11. Imprimir los primeros n términos de la serie de Fibonacci

```
n = int(input("Términos: "))
a, b = 0, 1
for _ in range(n):
    print(a, end=' ')
    a, b = b, a + b
```

Explicación: Cada número es la suma de los dos anteriores.

---

12. Pedir números hasta que se ingrese uno negativo, luego mostrar el promedio

```
suma = 0
contador = 0
while True:
    n = int(input("Número: "))
    if n < 0:
        break
    suma += n
    contador += 1
if contador > 0:
```

```
    print("Promedio:", suma / contador)
else:
    print("No se ingresaron números válidos")
```

Explicación: Se termina con un número negativo y se calcula el promedio.

---

13. Imprimir los divisores de un número

```
n = int(input("Número: "))
for i in range(1, n + 1):
    if n % i == 0:
        print(i)
```

Explicación: Se imprime cada divisor.

---

14. Simular 10 lanzamientos de dado y contar cuántas veces salió 6

```
import random
contador = 0
for _ in range(10):
    dado = random.randint(1, 6)
    print("Dado:", dado)
    if dado == 6:
        contador += 1
print("Cantidad de veces que salió 6:", contador)
```

Explicación: Se usa random.randint para simular un dado.

---

15. Mostrar una tabla de multiplicar completa del 1 al 10

```
for i in range(1, 11):
    for j in range(1, 11):
        print(f"{i} x {j} = {i * j}")
    print()
```

Explicación: Dos bucles anidados recorren las filas y columnas.

---

16. Detectar la primera letra duplicada en una palabra

```
palabra = input("Palabra: ")
vistas = set()
for letra in palabra:
    if letra in vistas:
        print("Primera repetida:", letra)
```

```
        break
    vistas.add(letra)
```

Explicación: Usa un conjunto para rastrear letras vistas.

---

17. Calcular el producto de los dígitos de un número

```
n = int(input("Número: "))
producto = 1
while n > 0:
    producto *= n % 10
    n //= 10
print("Producto de dígitos:", producto)
```

Explicación: Se multiplican los dígitos uno a uno.

---

18. Contar cuántos caracteres numéricos hay en una cadena

```
cadena = input("Cadena: ")
contador = 0
for c in cadena:
    if c.isdigit():
        contador += 1
print("Cantidad de dígitos:", contador)
```

Explicación: isdigit() identifica números.

---

19. Calcular cuántos números del 1 al 100 son múltiplos de 3 y 5

```
contador = 0
for i in range(1, 101):
    if i % 3 == 0 and i % 5 == 0:
        contador += 1
print("Múltiplos de 3 y 5:", contador)
```

Explicación: Se verifica si un número es divisible entre 3 y 5 a la vez.

---

20. Imprimir un triángulo rectángulo invertido de n filas

```
n = int(input("Filas: "))
for i in range(n, 0, -1):
    print("*" * i)
```

Dificultad avanzada

1 Números narcisistas de 3 cifras

Enunciado: Encuentra todos los números de 3 cifras que sean iguales a la suma de los cubos de sus dígitos.

Explicación: Un número narcisista de 3 cifras cumple: abc = $a^3 + b^3 + c^3$

Solución:

```
for num in range(100, 1000):
    a = num // 100
    b = (num // 10) % 10
    c = num % 10
    if num == a**3 + b**3 + c**3:
        print(num)
```

---

2. Pirámide de números centrada

Enunciado: Imprime una pirámide de altura n con números consecutivos centrados.

Ejemplo para n = 4:

```
   1
  2 3
 4 5 6
7 8 9 10
```

Solución:

```
n = 4
num = 1
for i in range(1, n + 1):
    print(' ' * (n - i), end='')
    for j in range(i):
        print(f'{num} ', end='')
        num += 1
    print()
```

---

3. Serie de Collatz

Enunciado: Dado un número n, imprime la serie de Collatz hasta llegar a 1.

Regla:

- Si n es par → n / 2
- Si n es impar → 3n + 1

Solución:

```
n = 13
while n != 1:
    print(n, end=' → ')
    n = n // 2 if n % 2 == 0 else 3 * n + 1
print(1)
```

---

4. Triángulo de Pascal

Enunciado: Imprime los primeros n niveles del triángulo de Pascal.

Solución:

```
n = 6
for i in range(n):
    num = 1
    print(' ' * (n - i), end='')
    for j in range(i + 1):
        print(f'{num} ', end='')
        num = num * (i - j) // (j + 1)
    print()
```

---

5. Primos gemelos

Enunciado: Imprime los primos gemelos entre 1 y 1000.

Explicación: Son pares de primos que se diferencian en 2 (ej. 11 y 13).

Solución:

```
def es_primo(n):
    if n < 2: return False
    for i in range(2, int(n**0.5)+1):
```

```
        if n % i == 0: return False
    return True

for i in range(2, 1000):
    if es_primo(i) and es_primo(i + 2):
        print(f'({i}, {i + 2})')
```

---

6. Palíndromos numéricos de 5 cifras

Enunciado: Encuentra todos los números palíndromos de 5 cifras.

Solución:

```
for n in range(10000, 100000):
    if str(n) == str(n)[::-1]:
        print(n)
```

---

7. Suma de dígitos hasta un solo dígito

Enunciado: Dado un número n, suma sus dígitos repetidamente hasta obtener un solo dígito.

Solución:

```
n = 9875
while n >= 10:
    n = sum(int(d) for d in str(n))
print(n)
```

---

8. Cuadrado mágico impar

Enunciado: Genera un cuadrado mágico de orden impar n.

Solución (método de Siam):

```
n = 5
matriz = [[0]*n for _ in range(n)]
i, j = 0, n//2

for num in range(1, n*n + 1):
    matriz[i][j] = num
```

```
    ni, nj = (i - 1) % n, (j + 1) % n
    if matriz[ni][nj]:
        i = (i + 1) % n
    else:
        i, j = ni, nj

for fila in matriz:
    print(fila)
```

---

9. Fracción continua

Enunciado: Calcula el valor de 1 + 1/(1 + 1/(1 + ...)) con n niveles.

Solución:

```
n = 10
res = 1
for _ in range(n):
    res = 1 + 1 / res
print(res)
```

---

10. Serie de Fibonacci inversa

Enunciado: Imprime la serie de Fibonacci de n elementos en orden inverso.

Solución:

```
n = 10
fib = [0, 1]
for i in range(2, n):
    fib.append(fib[-1] + fib[-2])
print(fib[::-1])
```

---

11. Factorización prima

Enunciado: Dado un número, imprime sus factores primos.

Solución:

```
n = 84
```

```
i = 2
while i <= n:
    if n % i == 0:
        print(i, end=' ')
        n //= i
    else:
        i += 1
```

---

12. Triángulo numérico creciente

Enunciado: Imprime un triángulo con números que se incrementan por línea.

Ejemplo para n = 4:

```
1
2 3
4 5 6
7 8 9 10
```

Solución:

```
n = 4
num = 1
for i in range(1, n + 1):
    for _ in range(i):
        print(num, end=' ')
        num += 1
    print()
```

---

13. Número más frecuente en una lista

Enunciado: Dada una lista, encuentra el número que más se repite (sin usar Counter).

Solución:

```
lista = [3, 2, 1, 3, 2, 3, 4, 1, 2, 3]
max_count = 0
for num in set(lista):
    count = 0
```

```
    for x in lista:
        if x == num:
            count += 1
    if count > max_count:
        max_count = count
        result = num
print(result)
```

---

14. Dibujar espiral cuadrada

Enunciado: Genera una matriz n x n con números en forma de espiral.

Solución:

```
n = 5
matriz = [[0]*n for _ in range(n)]
dx, dy = 0, 1
x = y = 0
for i in range(1, n*n + 1):
    matriz[x][y] = i
    nx, ny = x + dx, y + dy
    if not (0 <= nx < n and 0 <= ny < n and matriz[nx][ny] == 0):
        dx, dy = dy, -dx
    x += dx
    y += dy

for fila in matriz:
    print(fila)
```

---

15. Detectar ciclo en una secuencia

Enunciado: Dada una lista, detecta si hay un ciclo (una subsecuencia que se repite).

Solución:

```
lst = [1, 2, 3, 1, 2, 3, 1, 2, 3]
for size in range(1, len(lst)//2 + 1):
```

```
    pattern = lst[:size]
    repeated = pattern * (len(lst) // size)
    if repeated == lst[:len(repeated)]:
        print("Ciclo encontrado:", pattern)
        break
```

---

16. Palabras en forma de cruz

Enunciado: Dada una palabra, imprímela en forma de cruz.

Ejemplo:

c

a

s

a

s

a

Solución:

```
palabra = "casa"
print('\n'.join(palabra + palabra[-2::-1]))
```

---

17. Dibujar tabla de multiplicar

Enunciado: Imprime una tabla de multiplicar n x n.

Solución:

```
n = 10
for i in range(1, n + 1):
    for j in range(1, n + 1):
        print(f'{i * j:3}', end=' ')
    print()
```

---

18. Número más cercano a un promedio

Enunciado: Dada una lista, encuentra el número más cercano a su media.

Solución:

```
lista = [4, 9, 1, 7, 3]
prom = sum(lista) / len(lista)
cercano = lista[0]
for num in lista:
    if abs(num - prom) < abs(cercano - prom):
        cercano = num
print(cercano)
```

---

19. Verificar número perfecto

Enunciado: Comprueba si un número es perfecto (suma de divisores propios).

Solución:

```
n = 28
suma = 0
for i in range(1, n):
    if n % i == 0:
        suma += i
print(suma == n)
```

---

20. Crear reloj digital

Enunciado: Imprime todas las combinaciones posibles de un reloj de 24h (HH:MM:SS).

Solución:

```
for h in range(24):
    for m in range(60):
        for s in range(60):
            print(f'{h:02}:{m:02}:{s:02}')
```

---

4. Estructuras de datos, listas

Dificultad facil

---

1. Crear una lista con números del 1 al 5

Enunciado: Crea una lista que contenga los números del 1 al 5.

Solución:

numeros = [1, 2, 3, 4, 5]

print(numeros)

Explicación: Creamos una lista directamente con los valores deseados.

---

2. Acceder al segundo elemento de una lista

Enunciado: Dada la lista colores = ['rojo', 'verde', 'azul'], imprime el segundo color.

Solución:

colores = ['rojo', 'verde', 'azul']

print(colores[1])

Explicación: El índice 1 corresponde al segundo elemento de la lista.

---

3. Cambiar un valor en una lista

Enunciado: Cambia el valor 'verde' por 'amarillo' en la lista anterior.

Solución:

colores[1] = 'amarillo'

print(colores)

Explicación: Reasignamos el valor en la posición deseada usando su índice.

---

4. Agregar un elemento al final de una lista

Enunciado: Añade 'naranja' al final de la lista colores.

Solución:

colores.append('naranja')

print(colores)

Explicación: El método .append() agrega elementos al final.

---

5. Insertar un elemento en una posición específica

Enunciado: Inserta 'morado' en la segunda posición de la lista colores.

Solución:

colores.insert(1, 'morado')

print(colores)

Explicación: .insert(posición, valor) añade un elemento en la posición deseada.

---

6. Eliminar un elemento por su valor

Enunciado: Elimina 'rojo' de la lista.

Solución:

colores.remove('rojo')

print(colores)

Explicación: .remove() elimina el primer valor que coincide con el argumento.

---

7. Eliminar un elemento por su índice

Enunciado: Elimina el primer elemento de la lista colores.

Solución:

del colores[0]

print(colores)

Explicación: del elimina el elemento según su índice.

---

8. Obtener la longitud de una lista

Enunciado: Muestra cuántos elementos tiene la lista colores.

Solución:

print(len(colores))

Explicación: len(lista) devuelve la cantidad de elementos.

---

9. Ordenar una lista de números

Enunciado: Ordena la lista [5, 2, 4, 1, 3] de menor a mayor.

Solución:

numeros = [5, 2, 4, 1, 3]

numeros.sort()

print(numeros)

Explicación: .sort() ordena la lista *in place* (sin crear una nueva).

---

10. Ordenar al revés

Enunciado: Ordena la lista anterior de mayor a menor.

Solución:

numeros.sort(reverse=True)

print(numeros)

Explicación: reverse=True invierte el orden del .sort().

---

11. Verificar si un valor está en la lista

Enunciado: Comprueba si el número 3 está en la lista [1, 2, 3, 4, 5].

Solución:

print(3 in [1, 2, 3, 4, 5])

Explicación: in devuelve True o False si el valor está o no en la lista.

---

12. Sumar todos los elementos de una lista

Enunciado: Suma los elementos de la lista [1, 2, 3, 4, 5].

Solución:

numeros = [1, 2, 3, 4, 5]

print(sum(numeros))

Explicación: sum(lista) devuelve la suma de todos sus elementos.

---

13. Contar cuántas veces aparece un valor

Enunciado: Cuenta cuántas veces aparece el número 2 en [1, 2, 2, 3, 4].

Solución:

lista = [1, 2, 2, 3, 4]

print(lista.count(2))

Explicación: .count(valor) cuenta las apariciones del valor en la lista.

---

14. Copiar una lista

Enunciado: Crea una copia de la lista [1, 2, 3].

Solución:

```
original = [1, 2, 3]
copia = original.copy()
print(copia)
```

Explicación: .copy() crea una nueva lista con los mismos elementos.

---

15. Concatenar dos listas

Enunciado: Une las listas [1, 2] y [3, 4].

Solución:

```
lista1 = [1, 2]
lista2 = [3, 4]
resultado = lista1 + lista2
print(resultado)
```

Explicación: El operador + concatena listas.

---

16. Repetir una lista

Enunciado: Repite la lista [1, 2] tres veces.

Solución:

```
print([1, 2] * 3)
```

Explicación: El operador * repite los elementos de la lista.

---

17. Recorrer una lista con un bucle for

Enunciado: Imprime cada elemento de la lista [1, 2, 3].

Solución:

```
for numero in [1, 2, 3]:
    print(numero)
```

Explicación: El bucle for itera sobre cada elemento de la lista.

---

18. Obtener el índice de un elemento

Enunciado: Encuentra el índice del número 3 en [1, 2, 3, 4].

Solución:

```
lista = [1, 2, 3, 4]
print(lista.index(3))
```

Explicación: .index(valor) devuelve la posición del primer valor encontrado.

---

19. Vaciar una lista

Enunciado: Borra todos los elementos de la lista [1, 2, 3].

Solución:

```
lista = [1, 2, 3]
lista.clear()
print(lista)
```

Explicación: .clear() elimina todos los elementos de la lista.

---

20. Convertir una cadena en lista

Enunciado: Convierte la cadena "hola" en una lista de caracteres.

Solución:

```
palabra = "hola"
lista = list(palabra)
print(lista)
```

Explicación: list(cadena) convierte cada carácter en un elemento de la lista.

Dificultad media

1. Sumar elementos en posiciones pares

Enunciado: Dada una lista de enteros, suma los elementos que están en posiciones pares.

```
numeros = [4, 7, 2, 9, 5, 1]
```

Explicación: Se suman los elementos en índices 0, 2, 4.

Solución:

```
suma = sum(numeros[i] for i in range(0, len(numeros), 2))
print(suma)  # Resultado: 11
```

---

2. Eliminar duplicados

Enunciado: Elimina los elementos duplicados de una lista manteniendo el orden.

```
datos = [3, 5, 3, 7, 5, 9]
```

Solución:

```
resultado = []
for x in datos:
    if x not in resultado:
        resultado.append(x)
print(resultado)  # [3, 5, 7, 9]
```

---

3. Encontrar el segundo mayor

Enunciado: Encuentra el segundo número más grande en una lista.

```
valores = [10, 20, 4, 45, 99, 99]
```

Solución:

```
unicos = list(set(valores))
unicos.sort(reverse=True)
print(unicos[1])  # 45
```

---

4. Invertir lista sin usar reverse()

Enunciado: Invierte los elementos de una lista sin usar reverse() ni slicing.

```
lista = [1, 2, 3, 4]
```

Solución:

```
invertida = []
for i in range(len(lista)-1, -1, -1):
    invertida.append(lista[i])
print(invertida)  # [4, 3, 2, 1]
```

---

5. Separar pares e impares

Enunciado: Crea dos listas a partir de una original: una con los pares y otra con los impares.

nums = [2, 3, 8, 5, 6, 7]

Solución:

```
pares = [x for x in nums if x % 2 == 0]
impares = [x for x in nums if x % 2 != 0]
print(pares)   # [2, 8, 6]
print(impares) # [3, 5, 7]
```

---

6. Encontrar sublistas vacías

Enunciado: Cuenta cuántas sublistas vacías hay en una lista.

listas = [[], [1], [], [2, 3], [], []]

Solución:

```
vacias = sum(1 for sub in listas if sub == [])
print(vacias)  # 4
```

---

7. Comprobar si dos listas tienen algún elemento en común

a = [1, 2, 3]

b = [4, 5, 3]

Solución:

```
comun = any(x in b for x in a)
print(comun)  # True
```

---

8. Mover ceros al final

entrada = [0, 1, 0, 3, 12]

Solución:

```
no_ceros = [x for x in entrada if x != 0]
ceros = [0] * (len(entrada) - len(no_ceros))
print(no_ceros + ceros)  # [1, 3, 12, 0, 0]
```

---

9. Aplanar una lista de listas

listas = [[1, 2], [3, 4], [5]]

Solución:

```
resultado = [item for sublist in listas for item in sublist]
print(resultado)  # [1, 2, 3, 4, 5]
```

---

10. Duplicar elementos

original = [1, 2, 3]

Solución:

```
duplicada = [x * 2 for x in original]
print(duplicada)  # [2, 4, 6]
```

---

11. Intercambiar el primer y último elemento

lista = [10, 20, 30, 40]

Solución:

```
lista[0], lista[-1] = lista[-1], lista[0]
print(lista)  # [40, 20, 30, 10]
```

---

12. Lista sin extremos

lista = [5, 1, 8, 2, 9]

Solución:

```
nueva = lista[1:-1]
print(nueva)  # [1, 8, 2]
```

---

13. Lista de cuadrados

numeros = [1, 2, 3, 4]

Solución:

```
cuadrados = [x**2 for x in numeros]
print(cuadrados)  # [1, 4, 9, 16]
```

---

14. Contar ocurrencias de un número

lista = [1, 2, 3, 2, 2, 4]

Contar cuántas veces aparece el 2:

print(lista.count(2)) # 3

---

15. Insertar un elemento en el medio

lista = [1, 2, 4, 5]

Insertar 3 en el índice 2:

lista.insert(2, 3)

print(lista) # [1, 2, 3, 4, 5]

---

16. Combinar dos listas elemento a elemento

a = [1, 2, 3]

b = [4, 5, 6]

Solución:

combinada = [x + y for x, y in zip(a, b)]

print(combinada) # [5, 7, 9]

---

17. Encontrar elementos únicos en una lista

lista = [1, 2, 2, 3, 4, 4, 5]

Solución:

unicos = [x for x in lista if lista.count(x) == 1]

print(unicos) # [1, 3, 5]

---

18. Lista de booleanos según condición

edades = [18, 25, 17, 20]

¿Es mayor de edad?:

mayores = [edad >= 18 for edad in edades]

print(mayores) # [True, True, False, True]

---

19. Rotar lista hacia la izquierda

lista = [1, 2, 3, 4]

Rotar una posición:

rotada = lista[1:] + [lista[0]]

print(rotada) # [2, 3, 4, 1]

---

20. Eliminar elementos por valor

valores = [1, 2, 3, 2, 4]

Eliminar todos los 2:

filtrada = [x for x in valores if x != 2]

print(filtrada) # [1, 3, 4]

Dificultad avanzada

---

1. Invertir sublistas en una lista anidada

Enunciado: Dada una lista anidada, invierte cada sublista sin usar bucles explícitos.

lista = [[1, 2], [3, 4], [5, 6]]

Explicación: Queremos invertir cada sublista, no la lista principal.

Solución:

lista = [[1, 2], [3, 4], [5, 6]]

resultado = [sub[::-1] for sub in lista]

print(resultado) # [[2, 1], [4, 3], [6, 5]]

---

2. Eliminar duplicados preservando el orden

Enunciado: Dada una lista, elimina duplicados preservando el orden original.

datos = [3, 5, 2, 3, 5, 6, 2]

Solución:

datos = [3, 5, 2, 3, 5, 6, 2]

resultado = []

```
[resultado.append(x) for x in datos if x not in resultado]
print(resultado)  # [3, 5, 2, 6]
```

---

3. Rotar lista a la derecha n veces

Enunciado: Rota los elementos de la lista n veces a la derecha.

```
lista = [1, 2, 3, 4, 5]
n = 2
```

Solución:

```
lista = [1, 2, 3, 4, 5]
n = 2
n = n % len(lista)
rotada = lista[-n:] + lista[:-n]
print(rotada)  # [4, 5, 1, 2, 3]
```

---

4. Encontrar intersección sin usar conjuntos

Enunciado: Encuentra los elementos comunes entre dos listas sin usar set.

```
a = [1, 2, 3, 4]
b = [3, 4, 5, 6]
```

Solución:

```
interseccion = [x for x in a if x in b]
print(interseccion)  # [3, 4]
```

---

5. Lista de acumulados

Enunciado: Crea una lista donde cada elemento sea la suma acumulada.

```
entrada = [1, 2, 3, 4]
```

Solución:

```
acumulado = []
total = 0
for num in entrada:
    total += num
    acumulado.append(total)
```

```
print(acumulado)  # [1, 3, 6, 10]
```

---

6. Eliminar todos los elementos que aparezcan más de una vez

Enunciado: Elimina de la lista los elementos que aparecen más de una vez.

```
lista = [1, 2, 2, 3, 4, 4, 5]
```

Solución:

```
resultado = [x for x in lista if lista.count(x) == 1]
print(resultado)  # [1, 3, 5]
```

---

7. Transformar lista de tuplas a lista de listas

Enunciado: Convierte [(1,2), (3,4)] en [[1,2],[3,4]].

Solución:

```
tuplas = [(1, 2), (3, 4)]
listas = [list(t) for t in tuplas]
print(listas)  # [[1, 2], [3, 4]]
```

---

8. Flatten (aplanar) lista anidada de profundidad 2

Enunciado: Convierte [[1, 2], [3], [4, 5]] en [1, 2, 3, 4, 5].

Solución:

```
anidada = [[1, 2], [3], [4, 5]]
aplanada = [elem for sub in anidada for elem in sub]
print(aplanada)  # [1, 2, 3, 4, 5]
```

---

9. Encontrar todos los pares únicos con suma igual a n

Enunciado: Devuelve todos los pares únicos cuya suma sea n.

```
lista = [1, 3, 2, 2, 4]
n = 4
```

Solución:

```
pares = []
usados = set()
for i in lista:
```

```
    complemento = n - i
    if complemento in usados:
        par = tuple(sorted((i, complemento)))
        if par not in pares:
            pares.append(par)
    usados.add(i)
print(pares)  # [(1, 3), (2, 2)]
```

---

10. Ordenar lista por la suma de sus sublistas

Enunciado: Ordena por la suma de sus elementos.

```
datos = [[3, 1], [2, 2], [1, 1, 1]]
```

Solución:

```
ordenada = sorted(datos, key=sum)
print(ordenada)  # [[1, 1, 1], [2, 2], [3, 1]]
```

---

11. Crear matriz identidad de n x n

Enunciado: Crear matriz identidad con listas anidadas.

```
n = 4
```

Solución:

```
identidad = [[1 if i == j else 0 for j in range(n)] for i in range(n)]
print(identidad)
```

---

12. Encontrar el segundo mayor número

Enunciado: Encuentra el segundo mayor sin usar sort().

```
lista = [5, 2, 9, 1, 5]
```

Solución:

```
primero = segundo = float('-inf')
for x in lista:
    if x > primero:
        segundo, primero = primero, x
    elif primero > x > segundo:
```

```
    segundo = x
print(segundo)  # 5
```

---

13. Zipear listas de distinta longitud

Enunciado: Une [1,2,3] y [4,5] como [(1,4), (2,5), (3,None)].

Solución:

```
from itertools import zip_longest

a = [1, 2, 3]
b = [4, 5]
zipped = list(zip_longest(a, b))
print(zipped)  # [(1, 4), (2, 5), (3, None)]
```

---

14. Encontrar elementos que aparecen exactamente 2 veces

Enunciado: Devuelve los elementos con aparición exacta de 2 veces.

```
lista = [1, 2, 2, 3, 3, 3, 4]
```

Solución:

```
resultado = [x for x in set(lista) if lista.count(x) == 2]
print(resultado)  # [2]
```

---

15. Agrupar por índice par e impar

Enunciado: Divide la lista en dos: índices pares e impares.

```
datos = [10, 20, 30, 40, 50]
```

Solución:

```
pares = datos[::2]
impares = datos[1::2]
print(pares)    # [10, 30, 50]
print(impares)  # [20, 40]
```

---

16. Reemplazar elementos con la media de sus vecinos

Enunciado: Reemplaza cada elemento con la media de sus vecinos (excepto los extremos).

lista = [10, 20, 30, 40, 50]

Solución:

```
nueva = [lista[0]] + [(lista[i-1] + lista[i+1]) // 2 for i in range(1, len(lista)-1)] + [lista[-1]]
print(nueva)  # [10, 20, 30, 40, 50]
```

---

17. Contar elementos únicos sin set

Enunciado: Cuenta cuántos valores únicos hay en la lista.

datos = [1, 2, 2, 3, 4, 4, 5]

Solución:

```
unicos = []
[unicos.append(x) for x in datos if x not in unicos]
print(len(unicos))  # 5
```

---

18. Detectar si una lista es palíndromo

Enunciado: Devuelve True si la lista es igual a su reversa.

datos = [1, 2, 3, 2, 1]

Solución:

```
es_palindromo = datos == datos[::-1]
print(es_palindromo)  # True
```

---

19. Particionar en sublistas de tamaño fijo

Enunciado: Divide [1,2,3,4,5,6] en sublistas de 2.

Solución:

```
lista = [1, 2, 3, 4, 5, 6]
tamaño = 2
partes = [lista[i:i+tamaño] for i in range(0, len(lista), tamaño)]
print(partes)  # [[1, 2], [3, 4], [5, 6]]
```

---

20. Intercalar dos listas

Enunciado: Intercala [1,2,3] y [4,5,6] → [1,4,2,5,3,6].

Solución:

```
a = [1, 2, 3]
b = [4, 5, 6]
mezcla = [x for par in zip(a, b) for x in par]
print(mezcla)  # [1, 4, 2, 5, 3, 6]
```

---

5. Estructura de conjuntos

Dificultad facil

---

1. Crear un conjunto desde una lista

Ejercicio: Convierte la lista [1, 2, 2, 3, 4, 4, 5] a un conjunto.

Explicación: El conjunto elimina duplicados automáticamente.

Solución:

```
lista = [1, 2, 2, 3, 4, 4, 5]
conjunto = set(lista)
print(conjunto)  # {1, 2, 3, 4, 5}
```

---

2. Añadir un elemento al conjunto

Ejercicio: Añade el número 6 al conjunto {1, 2, 3}.

Solución:

```
s = {1, 2, 3}
s.add(6)
print(s)  # {1, 2, 3, 6}
```

---

3. Eliminar un elemento del conjunto

Ejercicio: Elimina el 2 del conjunto {1, 2, 3}.

Solución:

```
s = {1, 2, 3}
s.remove(2)
```

```
print(s)  # {1, 3}
```

4. Comprobar si un elemento está en un conjunto

Ejercicio: Comprueba si el número 4 está en el conjunto {1, 2, 3, 4}.

Solución:

```
s = {1, 2, 3, 4}
print(4 in s)  # True
```

---

5. Unir dos conjuntos

Ejercicio: Une los conjuntos {1, 2} y {3, 4}.

Solución:

```
a = {1, 2}
b = {3, 4}
union = a | b
print(union)  # {1, 2, 3, 4}
```

---

6. Intersección de dos conjuntos

Ejercicio: Encuentra la intersección de {1, 2, 3} y {2, 3, 4}.

Solución:

```
a = {1, 2, 3}
b = {2, 3, 4}
inter = a & b
print(inter)  # {2, 3}
```

---

7. Diferencia de conjuntos

Ejercicio: Encuentra qué elementos están en {1, 2, 3} pero no en {2, 3}.

Solución:

```
a = {1, 2, 3}
b = {2, 3}
diff = a - b
print(diff)  # {1}
```

---

8. Diferencia simétrica

Ejercicio: Muestra los elementos que están en un conjunto o el otro, pero no en ambos.

Solución:

```
a = {1, 2, 3}
b = {2, 3, 4}
sym_diff = a ^ b
print(sym_diff)  # {1, 4}
```

---

9. Convertir un string en conjunto

Ejercicio: Convierte "banana" en un conjunto.

Solución:

```
palabra = "banana"
letras = set(palabra)
print(letras)  # {'b', 'a', 'n'}
```

---

10. Longitud de un conjunto

Ejercicio: Cuenta cuántos elementos únicos hay en {1, 2, 2, 3}.

Solución:

```
conjunto = set([1, 2, 2, 3])
print(len(conjunto))  # 3
```

---

11. Crear conjunto vacío

Ejercicio: Crea un conjunto vacío correctamente.

Solución:

```
vacio = set()
print(vacio)  # set()
```

---

12. Eliminar sin error

Ejercicio: Elimina el 10 de {1, 2, 3} sin que dé error si no existe.

Solución:

```
s = {1, 2, 3}
```

```
s.discard(10)
print(s)  # {1, 2, 3}
```

---

13. Copiar un conjunto

Ejercicio: Haz una copia del conjunto {1, 2, 3}.

Solución:

```
s = {1, 2, 3}
copia = s.copy()
print(copia)  # {1, 2, 3}
```

---

14. Borrar todos los elementos

Ejercicio: Vacía el conjunto {1, 2, 3}.

Solución:

```
s = {1, 2, 3}
s.clear()
print(s)  # set()
```

---

15. Subconjunto

Ejercicio: ¿Es {1, 2} subconjunto de {1, 2, 3}?

Solución:

```
a = {1, 2}
b = {1, 2, 3}
print(a.issubset(b))  # True
```

---

16. Superconjunto

Ejercicio: ¿Es {1, 2, 3} superconjunto de {1}?

Solución:

```
a = {1, 2, 3}
b = {1}
print(a.issuperset(b))  # True
```

---

17. Iterar un conjunto

Ejercicio: Imprime todos los elementos de {1, 2, 3} usando un bucle.

Solución:

```
s = {1, 2, 3}
for elem in s:
    print(elem)
```

---

18. Sumar elementos únicos de una lista

Ejercicio: Suma solo los números únicos de [1, 2, 2, 3, 3, 4].

Solución:

```
lista = [1, 2, 2, 3, 3, 4]
suma = sum(set(lista))
print(suma)  # 10
```

---

19. Comparar dos conjuntos

Ejercicio: Di si {1, 2} y {2, 1} son iguales.

Solución:

```
print({1, 2} == {2, 1})  # True
```

---

20. Usar conjunto para quitar duplicados de una lista

Ejercicio: Quita duplicados de [1, 2, 2, 3, 1].

Solución:

```
lista = [1, 2, 2, 3, 1]
sin_duplicados = list(set(lista))
print(sin_duplicados)  # [1, 2, 3] (el orden puede variar)
```

Dificultad media

---

1. Crear un conjunto sin duplicados a partir de una lista

Enunciado: Dada una lista de números con valores repetidos, crea un conjunto sin duplicados.

```
lista = [1, 2, 2, 3, 4, 4, 5]
```

Explicación: Convertir la lista a conjunto elimina los duplicados.

Solución:

```
conjunto = set(lista)
print(conjunto)  # {1, 2, 3, 4, 5}
```

---

2. Verificar si un conjunto es subconjunto de otro

Enunciado: Comprueba si `{2, 3}` es subconjunto de `{1, 2, 3, 4}`.

Explicación: Usamos el método `.issubset()` o el operador `<=`.

Solución:

```
a = {2, 3}
b = {1, 2, 3, 4}
print(a.issubset(b))  # True
```

---

3. Calcular la unión de dos conjuntos

Enunciado: Une los conjuntos `{1, 2}` y `{2, 3, 4}`.

Explicación: Se usa `.union()` o el operador `|`.

Solución:

```
a = {1, 2}
b = {2, 3, 4}
union = a | b
print(union)  # {1, 2, 3, 4}
```

---

4. Calcular la intersección de dos conjuntos

Enunciado: Encuentra los elementos comunes entre `{1, 2, 3}` y `{2, 3, 4}`.

Explicación: Usar `.intersection()` o el operador `&`.

Solución:

```
a = {1, 2, 3}
```

```
b = {2, 3, 4}
print(a & b)  # {2, 3}
```

---

5. Diferencia de conjuntos

Enunciado: Encuentra los elementos que están en `{1, 2, 3}` pero no en `{2, 4}`.

Explicación: Se usa `.difference()` o el operador `-`.

Solución:

```
a = {1, 2, 3}
b = {2, 4}
print(a - b)  # {1, 3}
```

---

6. Diferencia simétrica

Enunciado: Encuentra los elementos que están en uno u otro, pero no en ambos.

Explicación: Se usa `.symmetric_difference()` o el operador `^`.

Solución:

```
a = {1, 2, 3}
b = {2, 3, 4}
print(a ^ b)  # {1, 4}
```

---

7. Eliminar un elemento de un conjunto

Enunciado: Elimina el número 2 de `{1, 2, 3}`.

Explicación: Se puede usar `.remove()` o `.discard()`.

Solución:

```
s = {1, 2, 3}
s.discard(2)
print(s)  # {1, 3}
```

---

8. Verificar si dos conjuntos son disjuntos

Enunciado: Comprueba si `{1, 2}` y `{3, 4}` no tienen elementos en común.

Explicación: `.isdisjoint()` devuelve True si no hay intersección.

Solución:

```
a = {1, 2}
b = {3, 4}
print(a.isdisjoint(b))  # True
```

---

9. Congelar un conjunto

Enunciado: Convierte `{1, 2, 3}` en un conjunto inmutable.

Explicación: Usamos `frozenset()`.

Solución:

```
f = frozenset({1, 2, 3})
print(f)  # frozenset({1, 2, 3})
```

---

10. Iterar sobre un conjunto

Enunciado: Recorre e imprime cada elemento de `{10, 20, 30}`.

Explicación: Se puede usar un bucle `for`.

Solución:

```
s = {10, 20, 30}
for elem in s:
    print(elem)
```

---

11. Crear un conjunto por comprensión

Enunciado: Crea un conjunto de cuadrados de los números del 1 al 5.

Explicación: Usa set comprehension.

Solución:

```
cuadrados = {x**2 for x in range(1, 6)}
print(cuadrados)  # {1, 4, 9, 16, 25}
```

---

12. Eliminar todos los elementos de un conjunto

Enunciado: Vacía el conjunto `{1, 2, 3}`.

Explicación: Usamos `.clear()`.

Solución:

```
s = {1, 2, 3}
s.clear()
print(s)  # set()
```

---

13. Tamaño de un conjunto

Enunciado: Calcula cuántos elementos tiene `{1, 2, 3, 4}`.

Explicación: Se usa `len()`.

Solución:

```
s = {1, 2, 3, 4}
print(len(s))  # 4
```

---

14. Añadir elementos de una lista a un conjunto

Enunciado: Agrega los elementos `[5, 6]` al conjunto `{1, 2}`.

Explicación: Usar `.update()` para añadir múltiples elementos.

Solución:

```
s = {1, 2}
s.update([5, 6])
print(s)  # {1, 2, 5, 6}
```

---

15. Convertir cadena en conjunto de caracteres únicos

Enunciado: De la cadena `"banana"`, crea un conjunto de letras únicas.

Explicación: Al convertir la cadena en `set`, se eliminan duplicados.

Solución:

```
palabra = "banana"
letras = set(palabra)
print(letras)  # {'a', 'b', 'n'}
```

---

16. Comparar si dos conjuntos son iguales

Enunciado: Verifica si `{1, 2, 3}` es igual a `{3, 2, 1}`.

Explicación: El orden no importa en los conjuntos.

Solución:

```
print({1, 2, 3} == {3, 2, 1})  # True
```

---

17. Quitar un elemento que podría no existir

Enunciado: Elimina 9 de `{1, 2, 3}` sin que dé error si no está.

Explicación: Usa `.discard()`, que no lanza excepción.

Solución:

```
s = {1, 2, 3}
s.discard(9)
print(s)  # {1, 2, 3}
```

---

18. Crear un conjunto de pares desde una lista

Enunciado: Crea un conjunto con los números pares de `[1, 2, 3, 4, 5, 6]`.

Explicación: Usa comprensión de conjuntos.

Solución:

```
lista = [1, 2, 3, 4, 5, 6]
pares = {x for x in lista if x % 2 == 0}
print(pares)  # {2, 4, 6}
```

---

19. Encontrar palabras comunes entre dos frases

Enunciado: Dadas dos frases, encuentra las palabras en común.

```
f1 = "el gato negro"
f2 = "el perro negro"
```

Explicación: Convertir a conjuntos y aplicar intersección.

Solución:

```
s1 = set(f1.split())
s2 = set(f2.split())
print(s1 & s2)  # {'el', 'negro'}
```

---

20. Quitar elementos de un conjunto usando otro

Enunciado: De `{1,2,3,4}`, quita los elementos de `{2,3}`.

Explicación: Se usa `.difference_update()`.

Solución:

```
a = {1, 2, 3, 4}
b = {2, 3}
a.difference_update(b)
print(a)  # {1, 4}
```

Dificultad avanzada

---

1. Encuentra los elementos únicos comunes en tres conjuntos

Enunciado:
Dado tres conjuntos A, B y C, encuentra los elementos que están presentes en los tres.

Explicación:
Utilizamos la intersección múltiple para obtener los elementos comunes.

Solución:

```
A = {1, 2, 3, 4}
B = {2, 3, 5, 6}
C = {3, 7, 2}

resultado = A & B & C
print(resultado)  # {2, 3}
```

---

2. Elimina duplicados manteniendo el orden

Enunciado:
Dada una lista con elementos repetidos, elimina duplicados usando un conjunto, pero conserva el orden.

Explicación:
Creamos un conjunto para ver si ya se ha visto un elemento.

Solución:

```
lista = [1, 2, 3, 2, 4, 1, 5]

vistos = set()

resultado = [x for x in lista if not (x in vistos or
vistos.add(x))]

print(resultado)  # [1, 2, 3, 4, 5]
```

---

3. Verifica si un conjunto es subconjunto propio de otro

Enunciado:
Crea una función que devuelva `True` si A es subconjunto **propio** de B.

Explicación:
Debe usar el operador **<** (subconjunto estricto).

Solución:

```
def es_subconjunto_propio(A, B):
    return A < B


print(es_subconjunto_propio({1, 2}, {1, 2, 3}))  # True
print(es_subconjunto_propio({1, 2, 3}, {1, 2, 3}))  # False
```

---

4. Diferencia simétrica recursiva de varios conjuntos

Enunciado:
Calcula la diferencia simétrica de varios conjuntos almacenados en una lista.

Explicación:
Usamos `functools.reduce` y `set.symmetric_difference`.

Solución:

```
from functools import reduce


conjuntos = [{1, 2, 3}, {2, 3, 4}, {3, 4, 5}]

resultado = reduce(set.symmetric_difference, conjuntos)

print(resultado)  # {1, 5}
```

---

5. Conjunto de palabras únicas de un texto

Enunciado:
Extrae todas las palabras únicas de un texto, ignorando mayúsculas.

Explicación:
Usamos `split()` y `lower()` y construimos un conjunto.

Solución:

```
texto = "Hola mundo hola Python Mundo"
palabras = set(texto.lower().split())
print(palabras)  # {'python', 'hola', 'mundo'}
```

---

6. Detecta conjuntos disjuntos en una lista

Enunciado:
Dada una lista de conjuntos, encuentra los pares disjuntos.

Explicación:
Un par de conjuntos es disjunto si su intersección es vacía.

Solución:

```
conjuntos = [{1, 2}, {3, 4}, {2, 5}]
for i in range(len(conjuntos)):
    for j in range(i + 1, len(conjuntos)):
        if conjuntos[i].isdisjoint(conjuntos[j]):
            print(f"{conjuntos[i]} y {conjuntos[j]} son
disjuntos")
```

---

7. Crea un conjunto inmutable de letras

Enunciado:
Usa `frozenset` para crear un conjunto inmutable de letras únicas.

Explicación:
Un `frozenset` no se puede modificar después de su creación.

Solución:

```
frase = "estructura"
inmutable = frozenset(frase)
print(inmutable)  # frozenset({'e', 's', 't', 'u', 'c', 'r', 'a'})
```

---

8. Intersección de todos los subconjuntos de una lista

Enunciado:
Halla la intersección de todos los conjuntos de una lista.

Explicación:
Usamos `reduce` con el operador `&`.

Solución:

```
from functools import reduce

lista_conjuntos = [{1, 2, 3}, {2, 3}, {2, 3, 4}]
comun = reduce(lambda a, b: a & b, lista_conjuntos)
print(comun)  # {2, 3}
```

---

9. Genera todas las combinaciones de pares posibles de un conjunto

Enunciado:
Dado un conjunto, genera todos los pares únicos posibles.

Explicación:
Usamos `itertools.combinations`.

Solución:

```
from itertools import combinations

s = {1, 2, 3, 4}
pares = list(combinations(s, 2))
print(pares)  # [(1, 2), (1, 3), (1, 4), ...]
```

---

10. Comprueba si dos listas contienen los mismos elementos sin importar el orden

Enunciado:
Dadas dos listas, verifica si tienen los mismos elementos únicos.

Explicación:
Convertimos ambas en conjuntos y los comparamos.

Solución:

```
def iguales(lista1, lista2):
    return set(lista1) == set(lista2)
```

```
print(iguales([1,2,3,2], [3,1,1,2]))  # True
```

---

11. Elimina todos los elementos comunes entre dos conjuntos

Enunciado:
Modifica A para que no tenga ningún elemento de B.

Explicación:
Usamos `A.difference_update(B)`.

Solución:

```
A = {1, 2, 3, 4}
B = {2, 3}
A.difference_update(B)
print(A)  # {1, 4}
```

---

12. Crea el conjunto potencia de un conjunto

Enunciado:
Genera el conjunto potencia (todas las combinaciones posibles).

Explicación:
Usamos `itertools.chain` y `combinations`.

Solución:

```
from itertools import chain, combinations

def conjunto_potencia(S):
    return list(chain.from_iterable(combinations(S, r) for r in range(len(S)+1)))

S = {1, 2, 3}
print(conjunto_potencia(S))  # [(), (1,), (2,), (3,), (1,2), ...]
```

---

13. Encuentra el primer elemento duplicado en una lista

Enunciado:
Devuelve el primer elemento que aparece más de una vez.

Explicación:
Usamos un conjunto para guardar los elementos ya vistos.

Solución:

```
def primer_duplicado(lista):
    vistos = set()
    for x in lista:
        if x in vistos:
            return x
        vistos.add(x)
    return None

print(primer_duplicado([1, 2, 3, 2, 4]))  # 2
```

---

14. Reemplaza elementos de una lista si están en un conjunto

Enunciado:
Cambia los valores de una lista a "X" si están en un conjunto prohibido.

Explicación:
Usamos comprensión de listas.

Solución:

```
prohibido = {3, 4}
lista = [1, 2, 3, 4, 5]
nueva = ["X" if x in prohibido else x for x in lista]
print(nueva)  # [1, 2, 'X', 'X', 5]
```

---

15. Comprueba si un conjunto es igual a su reverso (palíndromo lógico)

Enunciado:
Verifica si los elementos únicos de una lista son simétricos.

Explicación:
Convertimos en lista y comparamos con su reverso.

Solución:

```
def es_simetrico(lista):
    unicos = list(set(lista))
```

```
    return unicos == unicos[::-1]


print(es_simetrico([1,2,1]))  # True o False según el orden de set
(no garantiza orden)
```

---

16. Intersección múltiple sin usar operadores

Enunciado:
Halla la intersección de varios conjuntos sin usar &.

Explicación:
Utiliza `set.intersection()`.

Solución:

```
conjuntos = [{1,2,3}, {2,3,4}, {2,3,5}]

resultado = set.intersection(*conjuntos)

print(resultado)  # {2, 3}
```

---

17. Agrupa elementos por longitud con conjuntos

Enunciado:
Agrupa palabras por su longitud en un diccionario con valores tipo conjunto.

Explicación:
Usamos un diccionario con `setdefault`.

Solución:

```
palabras = ["uno", "dos", "cinco", "siete", "mil"]

agrupado = {}

for palabra in palabras:

    agrupado.setdefault(len(palabra), set()).add(palabra)

print(agrupado)  # {3: {'dos', 'uno', 'mil'}, 5: {'cinco',
'siete'}}
```

---

18. Crea un conjunto con números no repetidos entre dos listas

Enunciado:
Devuelve los elementos que solo aparecen una vez entre ambas listas.

Explicación:
Usamos `symmetric_difference`.

Solución:

```
A = [1,2,3,4]
B = [3,4,5,6]
resultado = set(A).symmetric_difference(B)
print(resultado)  # {1, 2, 5, 6}
```

---

19. Elimina elementos de un conjunto que cumplan condición

Enunciado:
Elimina todos los números pares de un conjunto.

Explicación:
Construimos un nuevo conjunto por comprensión.

Solución:

```
S = {1, 2, 3, 4, 5, 6}
S = {x for x in S if x % 2 != 0}
print(S)  # {1, 3, 5}
```

---

20. Verifica si dos conjuntos tienen la misma suma de elementos

Enunciado:
Compara la suma total de los elementos de dos conjuntos.

Explicación:
Usamos `sum()` sobre los conjuntos.

Solución:

```
A = {1, 2, 3}
B = {0, 3, 3}
print(sum(A) == sum(B))  # True
```

---

6. Estructura de datos, tuplas

Dificultad facil

---

1. Crear una tupla con 3 elementos

Enunciado:
Crea una tupla con los elementos `"manzana"`, `"banana"` y `"cereza"`.

Explicación:
Una tupla se define con paréntesis `()` y puede contener varios elementos.

Solución:

```
frutas = ("manzana", "banana", "cereza")
print(frutas)
```

---

2. Acceder al segundo elemento

Enunciado:
Dada la tupla `colores = ("rojo", "verde", "azul")`, imprime el segundo elemento.

Explicación:
Los índices en Python empiezan en 0. El segundo elemento tiene índice 1.

Solución:

```
colores = ("rojo", "verde", "azul")
print(colores[1])
```

---

3. Intentar modificar una tupla

Enunciado:
Intenta cambiar el primer valor de la tupla `dias = ("lunes", "martes", "miércoles")` a `"domingo"`.

Explicación:
Las tuplas son inmutables, no se pueden modificar.

Solución:

```
dias = ("lunes", "martes", "miércoles")
# Esto dará error:
# dias[0] = "domingo"
# print(dias)
# Resultado: TypeError
```

---

4. Concatenar dos tuplas

Enunciado:
Une las tuplas `(1, 2)` y `(3, 4)` en una sola.

Explicación:
Las tuplas se pueden concatenar usando el operador `+`.

Solución:

```
a = (1, 2)
b = (3, 4)
c = a + b
print(c)
```

---

5. Multiplicar una tupla

Enunciado:
Repite la tupla `("Hola",)` tres veces.

Explicación:
Puedes multiplicar tuplas con `*` para repetir sus elementos.

Solución:

```
saludo = ("Hola",) * 3
print(saludo)
```

---

6. Comprobar si un elemento está en una tupla

Enunciado:
Verifica si `"z"` está en la tupla `letras = ("x", "y", "z")`.

Explicación:
Se usa el operador `in` para verificar presencia.

Solución:

```
letras = ("x", "y", "z")
print("z" in letras)
```

---

7. Obtener la longitud de una tupla

Enunciado:
¿Cuántos elementos hay en la tupla `(1, 2, 3, 4)`?

Explicación:
La función `len()` devuelve el número de elementos.

Solución:

```
tupla = (1, 2, 3, 4)
print(len(tupla))
```

---

8. Convertir una lista en tupla

Enunciado:
Convierte la lista `["a", "b", "c"]` en una tupla.

Explicación:
Se usa la función `tuple()` para convertir listas a tuplas.

Solución:

```
lista = ["a", "b", "c"]
tupla = tuple(lista)
print(tupla)
```

---

9. Contar cuántas veces aparece un valor

Enunciado:
Cuenta cuántas veces aparece `5` en la tupla `(5, 1, 5, 3, 5)`.

Explicación:
Se usa el método `.count(valor)`.

Solución:

```
numeros = (5, 1, 5, 3, 5)
print(numeros.count(5))
```

---

10. Encontrar la posición de un elemento

Enunciado:
Encuentra el índice de `"verde"` en `("rojo", "verde", "azul")`.

Explicación:
El método `.index(valor)` devuelve la primera posición.

Solución:

```
colores = ("rojo", "verde", "azul")
```

```
print(colores.index("verde"))
```

---

11. Crear una tupla de un solo elemento

Enunciado:
Crea una tupla que contenga solo el número 7.

Explicación:
Una tupla de un solo elemento debe llevar una coma.

Solución:

```
tupla = (7,)
print(tupla)
```

---

12. Desempaquetar tupla en variables

Enunciado:
Desempaqueta la tupla `("Ana", 20)` en dos variables.

Explicación:
Puedes asignar directamente los elementos de la tupla a variables.

Solución:

```
persona = ("Ana", 20)
nombre, edad = persona
print(nombre)
print(edad)
```

---

13. Usar tupla como clave de diccionario

Enunciado:
Usa la tupla `(1, 2)` como clave en un diccionario.

Explicación:
Las tuplas se pueden usar como claves porque son inmutables.

Solución:

```
dic = {(1, 2): "punto"}
print(dic[(1, 2)])
```

---

14. Tupla dentro de una lista

Enunciado:
Crea una lista que contenga dos tuplas: `(1, 2)` y `(3, 4)`.

Explicación:
Puedes mezclar listas y tuplas sin problema.

Solución:

```
lista = [(1, 2), (3, 4)]
print(lista)
```

---

15. Tupla de tuplas

Enunciado:
Crea una tupla que contenga dos tuplas: `(5, 6)` y `(7, 8)`.

Explicación:
Una tupla puede contener otras tuplas.

Solución:

```
tupla = ((5, 6), (7, 8))
print(tupla)
```

---

16. Iterar sobre una tupla

Enunciado:
Imprime todos los elementos de la tupla `(10, 20, 30)`.

Explicación:
Puedes usar un `for` para recorrer sus elementos.

Solución:

```
tupla = (10, 20, 30)
for numero in tupla:
    print(numero)
```

---

17. Convertir tupla a string

Enunciado:
Convierte la tupla `("a", "b", "c")` en el string `"abc"`.

Explicación:
Usa `join()` para unir los elementos.

Solución:

```
letras = ("a", "b", "c")
resultado = "".join(letras)
print(resultado)
```

---

18. Tupla vacía

Enunciado:
Crea una tupla vacía.

Explicación:
Una tupla vacía no contiene ningún elemento.

Solución:

```
vacia = ()
print(vacia)
```

---

19. Verificar tipo de dato

Enunciado:
Confirma que `(1, 2)` es una tupla usando `type()`.

Explicación:
La función `type()` devuelve el tipo de un objeto.

Solución:

```
tupla = (1, 2)
print(type(tupla))
```

---

20. Sumar los valores de una tupla de enteros

Enunciado:
Suma todos los números de la tupla `(4, 5, 6)`.

Explicación:
Usa la función `sum()` para sumar elementos numéricos.

Solución:

```
numeros = (4, 5, 6)
print(sum(numeros))
```

Dificultad media

1. Invertir una tupla

Enunciado:
Invierte el orden de los elementos de la tupla `(10, 20, 30, 40)`.

Explicación:
Puedes usar slicing con paso `-1` para invertir la tupla.

Solución:

```
tupla = (10, 20, 30, 40)
invertida = tupla[::-1]
print(invertida)
```

---

2. Encontrar todos los índices de un valor

Enunciado:
Dada la tupla `(3, 5, 3, 3, 7)`, encuentra todos los índices donde aparece `3`.

Explicación:
Recorre con `enumerate()` para obtener índice y valor.

Solución:

```
tupla = (3, 5, 3, 3, 7)
indices = [i for i, x in enumerate(tupla) if x == 3]
print(indices)
```

---

3. Ordenar una tupla

Enunciado:
Ordena la tupla `(5, 2, 9, 1)` de menor a mayor.

Explicación:
Convierte en lista, ordena y vuelve a tupla.

Solución:

```
tupla = (5, 2, 9, 1)
ordenada = tuple(sorted(tupla))
print(ordenada)
```

---

4. Eliminar un elemento de una tupla

Enunciado:
Quita el número `3` de la tupla `(1, 2, 3, 4)`.

Explicación:
Las tuplas no se modifican, pero puedes crear una nueva sin el valor.

Solución:

```
tupla = (1, 2, 3, 4)
nueva = tuple(x for x in tupla if x != 3)
print(nueva)
```

---

5. Sumar elementos por posición entre dos tuplas

Enunciado:
Suma los elementos de `(1, 2, 3)` y `(4, 5, 6)` por posición.

Explicación:
Se usa `zip()` para sumar elementos en paralelo.

Solución:

```
a = (1, 2, 3)
b = (4, 5, 6)
resultado = tuple(x + y for x, y in zip(a, b))
print(resultado)
```

---

6. Crear una tupla con números pares de otra tupla

Enunciado:
Extrae los números pares de la tupla `(1, 2, 3, 4, 5, 6)`.

Explicación:
Usa una comprensión de tupla con condición.

Solución:

```
original = (1, 2, 3, 4, 5, 6)
pares = tuple(x for x in original if x % 2 == 0)
print(pares)
```

---

7. Contar elementos únicos en una tupla

Enunciado:
Cuenta cuántos elementos distintos hay en `(1, 2, 2, 3, 1, 4)`.

Explicación:
Convierte la tupla en un conjunto.

Solución:

```
tupla = (1, 2, 2, 3, 1, 4)
unicos = len(set(tupla))
print(unicos)
```

---

8. Agrupar elementos por pares

Enunciado:
Agrupa la tupla `(1, 2, 3, 4)` en pares: `((1,2), (3,4))`.

Explicación:
Requiere recorrer de dos en dos con slicing.

Solución:

```
tupla = (1, 2, 3, 4)
pares = tuple((tupla[i], tupla[i+1]) for i in range(0, len(tupla), 2))
print(pares)
```

---

9. Reemplazar un valor en una tupla

Enunciado:
Reemplaza el valor `3` por `9` en la tupla `(1, 2, 3, 4)`.

Explicación:
No puedes modificar, pero sí recrear la tupla con un reemplazo.

Solución:

```
tupla = (1, 2, 3, 4)
nueva = tuple(9 if x == 3 else x for x in tupla)
print(nueva)
```

---

10. Aplanar una tupla de tuplas

Enunciado:
Convierte `((1, 2), (3, 4))` en una tupla `(1, 2, 3, 4)`.

Explicación:
Se recorre con un doble `for`.

Solución:

```
tupla = ((1, 2), (3, 4))
aplanada = tuple(x for par in tupla for x in par)
print(aplanada)
```

---

11. Convertir una cadena en tupla de caracteres

Enunciado:
Convierte la cadena `"Python"` en una tupla de caracteres.

Explicación:
La función `tuple()` también funciona con strings.

Solución:

```
cadena = "Python"
tupla = tuple(cadena)
print(tupla)
```

---

12. Intercalar dos tuplas

Enunciado:
Intercala `(1, 3, 5)` y `(2, 4, 6)` → `(1, 2, 3, 4, 5, 6)`.

Explicación:
Se usa `zip()` y `sum()` para intercalar y aplanar.

Solución:

```
a = (1, 3, 5)
b = (2, 4, 6)
intercalada = tuple(sum(zip(a, b), ()))
print(intercalada)
```

---

13. Encontrar el valor máximo sin usar `max()`

Enunciado:
Halla el valor más alto en la tupla `(3, 8, 1, 9)` sin `max()`.

Explicación:
Recorre los elementos comparando.

Solución:

```
tupla = (3, 8, 1, 9)
mayor = tupla[0]
for x in tupla:
    if x > mayor:
        mayor = x
print(mayor)
```

---

14. Crear tupla con valores únicos ordenados

Enunciado:
De la tupla `(3, 1, 3, 2)`, obtener `(1, 2, 3)`.

Explicación:
Usa `set()` y `sorted()`.

Solución:

```
tupla = (3, 1, 3, 2)
resultado = tuple(sorted(set(tupla)))
print(resultado)
```

---

15. Convertir tupla de listas en lista de tuplas

Enunciado:
Convierte `([1, 2], [3, 4])` en `[(1, 2), (3, 4)]`.

Explicación:
Requiere recorrer y convertir cada sublista.

Solución:

```
entrada = ([1, 2], [3, 4])
salida = [tuple(x) for x in entrada]
print(salida)
```

---

16. Encontrar pares (a, b) con suma 10

Enunciado:
De la tupla `(1, 2, 3, 7, 8, 9)`, encuentra pares que sumen 10.

Explicación:
Busca combinaciones sin repetir.

Solución:

```
datos = (1, 2, 3, 7, 8, 9)

pares = [(a, b) for i, a in enumerate(datos) for b in datos[i+1:] if a + b == 10]

print(pares)
```

---

17. Contar elementos mayores que un número

Enunciado:
Cuenta cuántos valores en `(4, 6, 2, 9)` son mayores que `5`.

Explicación:
Filtra y cuenta.

Solución:

```
tupla = (4, 6, 2, 9)

contador = sum(1 for x in tupla if x > 5)

print(contador)
```

---

18. Eliminar duplicados y mantener orden

Enunciado:
De `(1, 2, 1, 3, 2)`, obtener `(1, 2, 3)` sin perder el orden.

Explicación:
Usa un conjunto auxiliar para registrar elementos ya vistos.

Solución:

```
tupla = (1, 2, 1, 3, 2)

vistos = set()

resultado = tuple(x for x in tupla if not (x in vistos or vistos.add(x)))

print(resultado)
```

---

19. Convertir lista de tuplas en diccionario

Enunciado:
Convierte `[(1, "a"), (2, "b")]` en `{1: "a", 2: "b"}`.

Explicación:
Usa `dict()` sobre una lista de pares.

Solución:

```
lista = [(1, "a"), (2, "b")]
diccionario = dict(lista)
print(diccionario)
```

---

20. Combinar tupla de claves y tupla de valores

Enunciado:
Con `(“a”, “b”)` y `(1, 2)`, forma `{“a”: 1, “b”: 2}`.

Explicación:
`zip()` une elementos por posición.

Solución:

```
claves = ("a", "b")
valores = (1, 2)
diccionario = dict(zip(claves, valores))
print(diccionario)
```

Nivel avanzado

---

1. Crear una tupla de 100 números pares y contar cuántos son divisibles por 10.
Explicación: Requiere generar una secuencia, crear una tupla, y filtrar por condición.
Solución:

```
numeros = tuple(range(0, 200, 2))
div10 = len([x for x in numeros if x % 10 == 0])
print(div10)  # Salida: 20
```

---

2. Dada una tupla de palabras, contar cuántas tienen más de 5 letras.
Explicación: Iterar y aplicar condición sobre cada elemento.
Solución:

```
palabras = ("Python", "estructura", "datos", "tupla", "avanzado", "nivel", "examen")

largas = len([p for p in palabras if len(p) > 5])

print(largas)  # Salida: 4
```

---

3. Comprobar si una tupla contiene solo valores únicos (sin repeticiones).
Explicación: Convertir a set y comparar longitudes.
Solución:

```
t = (1, 2, 3, 4, 5, 1)

es_unica = len(t) == len(set(t))

print(es_unica)  # Salida: False
```

---

4. Obtener los índices de todos los elementos 'a' en una tupla.
Explicación: Recorrer la tupla con `enumerate`.
Solución:

```
t = ('a', 'b', 'c', 'a', 'd', 'a')

indices = [i for i, x in enumerate(t) if x == 'a']

print(indices)  # Salida: [0, 3, 5]
```

---

5. Convertir una lista de tuplas (clave, valor) en un diccionario.
Explicación: Usar `dict()` constructor.
Solución:

```
pares = [("uno", 1), ("dos", 2), ("tres", 3)]

d = dict(pares)

print(d)  # Salida: {'uno': 1, 'dos': 2, 'tres': 3}
```

---

6. Ordenar una tupla de tuplas por el segundo valor.
Explicación: Utilizar `sorted()` con `key`.
Solución:

```
t = ((1, 4), (3, 2), (2, 5))
```

```
ordenado = tuple(sorted(t, key=lambda x: x[1]))
print(ordenado)  # Salida: ((3, 2), (1, 4), (2, 5))
```

---

7. Intercambiar claves y valores en una tupla de pares (clave, valor).
Explicación: Intercambio con comprensión.
Solución:

```
original = (("a", 1), ("b", 2), ("c", 3))
invertido = tuple((v, k) for k, v in original)
print(invertido)  # Salida: ((1, 'a'), (2, 'b'), (3, 'c'))
```

---

8. Aplanar una tupla de tuplas en una sola tupla.
Explicación: Iterar internamente por subtuplas.
Solución:

```
t = ((1, 2), (3, 4), (5, 6))
plana = tuple(x for sub in t for x in sub)
print(plana)  # Salida: (1, 2, 3, 4, 5, 6)
```

---

9. Eliminar duplicados de una tupla manteniendo el orden.
Explicación: Usar conjunto auxiliar para mantener únicos.
Solución:

```
t = (1, 2, 2, 3, 1, 4)
visto = set()
resultado = tuple(x for x in t if not (x in visto or
visto.add(x)))
print(resultado)  # Salida: (1, 2, 3, 4)
```

---

10. Sumar todas las posiciones pares de una tupla numérica.
Explicación: Acceder con slicing y `sum()`.
Solución:

```
t = (10, 20, 30, 40, 50, 60)
suma = sum(t[::2])
print(suma)  # Salida: 90
```

---

11. Crear una tupla con los cuadrados de los primeros 30 enteros.
Explicación: Uso de `range()` y comprensión de tupla.
Solución:

```
cuadrados = tuple(x**2 for x in range(30))
print(cuadrados)  # Salida: (0, 1, 4, ..., 841)
```

---

12. Encontrar el valor máximo y mínimo en una tupla.
Explicación: Usar funciones `max()` y `min()`.
Solución:

```
t = (5, 8, 2, 9, 4)
print(max(t), min(t))  # Salida: 9 2
```

---

13. Revertir una tupla usando slicing.
Explicación: `[::-1]` revierte la secuencia.
Solución:

```
t = (1, 2, 3, 4)
print(t[::-1])  # Salida: (4, 3, 2, 1)
```

---

14. Comprobar si dos tuplas son anagramas (mismos elementos en diferente orden).
Explicación: Comparar `sorted()`.
Solución:

```
a = (1, 2, 3)
b = (3, 1, 2)
son_anagramas = sorted(a) == sorted(b)
print(son_anagramas)  # Salida: True
```

---

15. Dividir una tupla grande en sub-tuplas de 3 elementos.
Explicación: Slicing por pasos.
Solución:

```
t = tuple(range(12))
partes = tuple(t[i:i+3] for i in range(0, len(t), 3))
print(partes)  # Salida: ((0, 1, 2), (3, 4, 5), ...)
```

---

16. Obtener la intersección de dos tuplas sin convertirlas en sets.
Explicación: Filtrado manual y eliminación de duplicados.
Solución:

```
a = (1, 2, 3, 4)
b = (3, 4, 5, 6)
inter = tuple(x for x in a if x in b)
print(inter)  # Salida: (3, 4)
```

---

17. Encontrar los 3 elementos más frecuentes en una tupla.
Explicación: Usar `collections.Counter` y `most_common`.
Solución:

```
from collections import Counter
t = (1, 2, 2, 3, 3, 3, 4, 4, 4, 4)
mas = tuple(x for x, _ in Counter(t).most_common(3))
print(mas)  # Salida: (4, 3, 2)
```

---

18. Reemplazar un valor específico en una tupla (crear nueva).
Explicación: Tuplas son inmutables, se crea otra.
Solución:

```
t = (1, 2, 3, 2, 4)
nuevo = tuple(99 if x == 2 else x for x in t)
print(nuevo)  # Salida: (1, 99, 3, 99, 4)
```

---

19. Verificar si una tupla contiene solo números enteros.
Explicación: Comprobar tipo de cada elemento.
Solución:

```
t = (1, 2, 3, "a")
solo_enteros = all(isinstance(x, int) for x in t)
print(solo_enteros)  # Salida: False
```

---

20. Generar todas las permutaciones posibles de una tupla de 3 elementos.
Explicación: Usar `itertools.permutations`.
Solución:

```
from itertools import permutations
```

```
t = (1, 2, 3)
perms = tuple(permutations(t))
print(perms)  # Salida: ((1, 2, 3), (1, 3, 2), ...)
```

---

7. Estructura de datos, diccionarios
Dificultad facil

---

1. Crear un diccionario simple

Enunciado:
Crea un diccionario que contenga el nombre y la edad de una persona.

Solución y explicación:

```
persona = {"nombre": "Ana", "edad": 30}
print(persona)
```

Creamos un diccionario con dos claves: `"nombre"` y `"edad"`, y les asignamos sus respectivos valores.

---

2. Acceder a un valor del diccionario

Enunciado:
Muestra por pantalla la edad de la persona del diccionario del ejercicio anterior.

Solución y explicación:

```
print(persona["edad"])
```

Accedemos al valor de la clave `"edad"` usando corchetes.

---

3. Modificar un valor

Enunciado:
Cambia la edad de Ana a 31.

Solución y explicación:

```
persona["edad"] = 31
print(persona)
```

Actualizamos el valor de una clave como si fuera una variable.

---

4. Añadir una nueva clave

Enunciado:
Agrega una nueva clave llamada `"ciudad"` con el valor `"Zaragoza"`.

Solución y explicación:

```
persona["ciudad"] = "Zaragoza"
print(persona)
```

Así se añade una clave nueva simplemente asignándole un valor.

---

5. Eliminar una clave

Enunciado:
Elimina la clave `"ciudad"` del diccionario.

Solución y explicación:

```
del persona["ciudad"]
print(persona)
```

`del` se usa para eliminar claves del diccionario.

---

6. Ver todas las claves

Enunciado:
Muestra todas las claves del diccionario `persona`.

Solución y explicación:

```
print(persona.keys())
```

`.keys()` devuelve una vista con todas las claves.

---

7. Ver todos los valores

Enunciado:
Muestra todos los valores del diccionario `persona`.

Solución y explicación:

```
print(persona.values())
```

`.values()` devuelve los valores del diccionario.

---

8. Ver todos los pares clave-valor

Enunciado:
Muestra todos los pares clave-valor del diccionario.

Solución y explicación:

```
print(persona.items())
```

`.items()` devuelve una vista de tuplas con cada clave y su valor.

---

9. Usar un bucle `for` con `items()`

Enunciado:
Recorre el diccionario e imprime cada clave con su valor.

Solución y explicación:

```
for clave, valor in persona.items():
    print(clave, ":", valor)
```

El método `.items()` permite iterar sobre pares clave-valor directamente.

---

10. Comprobar si una clave existe

Enunciado:
Comprueba si la clave `"nombre"` está en el diccionario.

Solución y explicación:

```
if "nombre" in persona:
    print("Sí, está.")
```

El operador `in` verifica si una clave existe en el diccionario.

---

11. Crear un diccionario vacío

Enunciado:
Crea un diccionario vacío llamado `datos`.

Solución y explicación:

```
datos = {}
print(datos)
```

Se usa `{}` o `dict()` para inicializar un diccionario vacío.

---

12. Diccionario de cuadrados

Enunciado:
Crea un diccionario con los números del 1 al 5 como claves y sus cuadrados como valores.

Solución y explicación:

```
cuadrados = {x: x**2 for x in range(1, 6)}
print(cuadrados)
```

Usamos un diccionario por comprensión para llenar automáticamente las claves y valores.

---

13. Contar letras de una palabra

Enunciado:
Cuenta cuántas veces aparece cada letra en la palabra `"banana"`.

Solución y explicación:

```
palabra = "banana"
conteo = {}
for letra in palabra:
    if letra in conteo:
        conteo[letra] += 1
    else:
        conteo[letra] = 1
print(conteo)
```

Usamos un diccionario para acumular la cantidad de veces que aparece cada letra.

---

14. Obtener valor con `.get()`

Enunciado:
Usa `.get()` para obtener la edad de una persona, y muestra "No encontrado" si no existe.

Solución y explicación:

```
print(persona.get("edad", "No encontrado"))
print(persona.get("altura", "No encontrado"))
```

`.get()` evita errores si la clave no está en el diccionario.

---

15. Copiar un diccionario

Enunciado:
Crea una copia del diccionario `persona` en otro llamado `persona2`.

Solución y explicación:

```
persona2 = persona.copy()
print(persona2)
```

El método `.copy()` crea una copia superficial del diccionario.

---

16. Borrar todo el diccionario

Enunciado:
Vacía por completo el diccionario `persona`.

Solución y explicación:

```
persona.clear()
print(persona)
```

`.clear()` elimina todos los elementos del diccionario.

---

17. Fusionar dos diccionarios

Enunciado:
Une los diccionarios `a = {"x":1}` y `b = {"y":2}` en uno solo.

Solución y explicación:

```
a = {"x":1}
b = {"y":2}
a.update(b)
print(a)
```

El método `.update()` agrega claves y valores de otro diccionario.

---

18. Número de elementos del diccionario

Enunciado:
Muestra cuántos pares clave-valor tiene el diccionario `a`.

Solución y explicación:

```
print(len(a))
```

`len()` aplicado a un diccionario devuelve la cantidad de claves.

---

19. Diccionario con listas

Enunciado:
Crea un diccionario que contenga listas como valores: `"colores": ["rojo", "azul"]`.

Solución y explicación:

```
colores = {"colores": ["rojo", "azul"]}
print(colores)
```

Un valor puede ser cualquier tipo de dato, incluida una lista.

---

20. Iterar sobre una lista de diccionarios

Enunciado:
Dado `personas = [{"nombre": "Ana"}, {"nombre": "Luis"}]`, imprime todos los nombres.

Solución y explicación:

```
personas = [{"nombre": "Ana"}, {"nombre": "Luis"}]
for p in personas:
    print(p["nombre"])
```

Cada elemento de la lista es un diccionario, así que se accede como siempre.

Dificultad media

---

1. Contar la frecuencia de cada carácter en una cadena

Enunciado:
Escribe un programa que cuente cuántas veces aparece cada carácter en una cadena dada.

Explicación:
Se recorre la cadena y se usa un diccionario para llevar la cuenta de cada carácter.

Solución:

```
cadena = "python estructura de datos"
frecuencia = {}
```

```
for char in cadena:
    if char != " ":
        frecuencia[char] = frecuencia.get(char, 0) + 1

print(frecuencia)
```

---

2. Sumar los valores de un diccionario

Enunciado:
Dado un diccionario con valores numéricos, calcula la suma de todos sus valores.

Explicación:
Se utiliza `sum()` sobre los valores del diccionario.

Solución:

```
datos = {'a': 10, 'b': 20, 'c': 30}
total = sum(datos.values())
print(total)  # 60
```

---

3. Invertir un diccionario

Enunciado:
Dado un diccionario, crea uno nuevo invirtiendo claves y valores.

Explicación:
Se recorre el diccionario original asignando `valor: clave` en el nuevo.

Solución:

```
original = {'a': 1, 'b': 2, 'c': 3}
invertido = {v: k for k, v in original.items()}
print(invertido)  # {1: 'a', 2: 'b', 3: 'c'}
```

---

4. Eliminar claves duplicadas por valor

Enunciado:
Elimina claves duplicadas si comparten el mismo valor.

Explicación:
Se usa un set para recordar los valores únicos.

Solución:

```
d = {'a': 1, 'b': 2, 'c': 1, 'd': 3}
```

```
nuevo = {}
valores = set()

for k, v in d.items():
    if v not in valores:
        nuevo[k] = v
        valores.add(v)

print(nuevo)
```

---

5. Combinar dos diccionarios sumando valores

Enunciado:
Dados dos diccionarios con claves comunes, combínalos sumando los valores de claves coincidentes.

Explicación:
Se recorre ambos diccionarios y se actualizan los valores sumados.

Solución:

```
d1 = {'a': 1, 'b': 2}
d2 = {'a': 3, 'b': 1, 'c': 5}

resultado = d1.copy()
for k, v in d2.items():
    resultado[k] = resultado.get(k, 0) + v

print(resultado)  # {'a': 4, 'b': 3, 'c': 5}
```

---

6. Filtrar un diccionario por valor

Enunciado:
Conserva solo las claves cuyo valor sea mayor a 10.

Explicación:
Se crea un nuevo diccionario filtrando por condición.

Solución:

```
d = {'a': 5, 'b': 15, 'c': 8, 'd': 20}
filtrado = {k: v for k, v in d.items() if v > 10}
print(filtrado)  # {'b': 15, 'd': 20}
```

---

7. Crear un diccionario desde dos listas

Enunciado:
Dadas dos listas de igual longitud, crea un diccionario con una como claves y otra como valores.

Explicación:
Se usa `zip()` para unir ambas listas.

Solución:

```
claves = ['a', 'b', 'c']
valores = [1, 2, 3]
diccionario = dict(zip(claves, valores))
print(diccionario)
```

---

8. Contar palabras en un texto

Enunciado:
Cuenta cuántas veces aparece cada palabra en un texto.

Explicación:
Se divide el texto en palabras y se cuentan con un diccionario.

Solución:

```
texto = "hola mundo hola python mundo"
palabras = texto.split()
conteo = {}

for palabra in palabras:
    conteo[palabra] = conteo.get(palabra, 0) + 1

print(conteo)
```

---

9. Ordenar diccionario por valor descendente

Enunciado:
Ordena un diccionario por sus valores de forma descendente.

Explicación:
Se usa `sorted()` con `key` sobre los valores.

Solución:

```
d = {'a': 3, 'b': 1, 'c': 5}
ordenado = dict(sorted(d.items(), key=lambda item: item[1],
reverse=True))
print(ordenado)
```

---

10. Agrupar elementos por longitud de palabra

Enunciado:
Agrupa palabras según su longitud en un diccionario.

Explicación:
Usa la longitud como clave.

Solución:

```
palabras = ['casa', 'sol', 'mar', 'perro', 'ratón']
grupo = {}

for p in palabras:
    l = len(p)
    grupo.setdefault(l, []).append(p)

print(grupo)
```

---

11. Diccionario con cuadrados de números

Enunciado:
Crea un diccionario con los números del 1 al 10 como claves y sus cuadrados como valores.

Explicación:
Usa `range()` y comprensión de diccionario.

Solución:

```
cuadrados = {i: i**2 for i in range(1, 11)}
print(cuadrados)
```

---

12. Verificar si dos diccionarios son iguales

Enunciado:
Comprueba si dos diccionarios tienen las mismas claves y valores.

Explicación:
Se usa el operador `==`.

Solución:

```
d1 = {'a': 1, 'b': 2}
d2 = {'b': 2, 'a': 1}
print(d1 == d2)  # True
```

---

13. Contar elementos únicos en valores

Enunciado:
Cuenta cuántos elementos únicos hay en los valores (listas) de un diccionario.

Explicación:
Convierte las listas en un conjunto.

Solución:

```
d = {'a': [1, 2], 'b': [2, 3], 'c': [1, 4]}
todos = set()

for lista in d.values():
    todos.update(lista)

print(len(todos))  # 4
```

---

14. Crear diccionario de letras y sus posiciones

Enunciado:
Dada una palabra, crea un diccionario que mapee cada letra a la lista de posiciones donde aparece.

Explicación:
Se usa un diccionario con listas.

Solución:

```
palabra = "banana"
posiciones = {}

for i, letra in enumerate(palabra):
    posiciones.setdefault(letra, []).append(i)

print(posiciones)
```

---

15. Convertir lista de tuplas a diccionario

Enunciado:
Convierte `[('a', 1), ('b', 2)]` en `{'a': 1, 'b': 2}`.

Explicación:
Se usa `dict()` directamente.

Solución:

```
tuplas = [('a', 1), ('b', 2)]
```

```
d = dict(tuplas)
print(d)
```

---

16. Contar vocales con diccionario

Enunciado:
Cuenta cuántas veces aparece cada vocal en una cadena.

Explicación:
Se recorre la cadena y se filtran las vocales.

Solución:

```
texto = "estructura de datos"
vocales = "aeiou"
conteo = {}

for letra in texto:
    if letra in vocales:
        conteo[letra] = conteo.get(letra, 0) + 1

print(conteo)
```

---

17. Eliminar claves con valor nulo

Enunciado:
Quita del diccionario las claves cuyo valor sea `None`.

Explicación:
Se filtra usando comprensión de diccionario.

Solución:

```
d = {'a': None, 'b': 2, 'c': None, 'd': 3}
limpio = {k: v for k, v in d.items() if v is not None}
print(limpio)
```

---

18. Fusionar diccionarios con prioridad al segundo

Enunciado:
Fusiona dos diccionarios dando prioridad a los valores del segundo.

Explicación:
Usa `update()` para sobreescribir.

Solución:

```
d1 = {'a': 1, 'b': 2}
d2 = {'b': 3, 'c': 4}
fusionado = d1.copy()
fusionado.update(d2)
print(fusionado)
```

---

19. Calcular promedio por estudiante

Enunciado:
Dado un diccionario con listas de notas, calcula el promedio por estudiante.

Explicación:
Se usa `sum()` y `len()` sobre cada lista.

Solución:

```
notas = {'Ana': [8, 9], 'Luis': [6, 7, 5]}
promedios = {alumno: sum(n) / len(n) for alumno, n in
notas.items()}
print(promedios)
```

---

20. Crear diccionario de frecuencia de dígitos

Enunciado:
Cuenta la frecuencia de los dígitos en una lista de números.

Explicación:
Convierte cada número a cadena para acceder a sus dígitos.

Solución:

```
numeros = [123, 345, 151, 213]
frecuencia = {}

for numero in numeros:
    for d in str(numero):
        frecuencia[d] = frecuencia.get(d, 0) + 1

print(frecuencia)
```

Dificultad avanzada

---

1. Contar ocurrencias únicas de valores en un diccionario

Enunciado:
Dado un diccionario que contiene nombres como claves y colores favoritos como valores, cuenta cuántas veces aparece cada color.

```
datos = {'Ana': 'rojo', 'Luis': 'azul', 'Eva': 'rojo', 'Carlos': 'verde', 'Sofía': 'azul'}
```

Explicación:
Usamos un diccionario auxiliar para contar los valores repetidos.

Solución:

```
from collections import Counter

conteo = Counter(datos.values())

print(conteo)
```

---

2. Invertir un diccionario

Enunciado:
Invierte las claves y valores de un diccionario. Si hay claves duplicadas, guarda los valores en una lista.

```
datos = {'a': 1, 'b': 2, 'c': 1}
```

Explicación:
Creamos un nuevo diccionario y agrupamos por valor original.

Solución:

```
invertido = {}

for k, v in datos.items():

    invertido.setdefault(v, []).append(k)

print(invertido)
```

---

3. Ordenar un diccionario por valores descendentes

Enunciado:
Ordena un diccionario según sus valores de mayor a menor.

```
puntos = {'A': 30, 'B': 10, 'C': 50}
```

Explicación:
Usamos `sorted` con `key` sobre `items()` y `reverse=True`.

Solución:

```
ordenado = dict(sorted(puntos.items(), key=lambda x: x[1],
reverse=True))
print(ordenado)
```

---

4. Sumar valores de claves comunes en dos diccionarios

Enunciado:
Suma los valores de dos diccionarios que tienen claves coincidentes.

```
d1 = {'a': 100, 'b': 200, 'c': 300}
d2 = {'a': 300, 'b': 100, 'd': 400}
```

Explicación:
Iteramos sobre claves de ambos diccionarios sumando los valores o usando `collections.Counter`.

Solución:

```
from collections import Counter
resultado = dict(Counter(d1) + Counter(d2))
print(resultado)
```

---

5. Filtrar un diccionario por valores mayores a un umbral

Enunciado:
Filtra los pares clave-valor con valores mayores a 50.

```
datos = {'a': 40, 'b': 60, 'c': 80, 'd': 30}
```

Explicación:
Usamos diccionario por comprensión.

Solución:

```
filtrado = {k: v for k, v in datos.items() if v > 50}
print(filtrado)
```

---

6. Fusionar varios diccionarios

Enunciado:
Fusiona una lista de diccionarios sumando claves repetidas.

```
lista = [{'a': 2, 'b': 3}, {'a': 5, 'c': 4}, {'b': 1, 'c': 2}]
```

Explicación:
Acumulamos los valores por clave.

Solución:

```
from collections import defaultdict
resultado = defaultdict(int)
for d in lista:
    for k, v in d.items():
        resultado[k] += v
print(dict(resultado))
```

---

7. Agrupar palabras por longitud

Enunciado:
Dada una lista de palabras, agrúpalas en un diccionario por su longitud.

```
palabras = ['sol', 'luna', 'mar', 'estrella']
```

Explicación:
Usamos la longitud como clave y listas como valores.

Solución:

```
from collections import defaultdict
agrupadas = defaultdict(list)
for palabra in palabras:
    agrupadas[len(palabra)].append(palabra)
print(dict(agrupadas))
```

---

8. Detectar claves duplicadas en varios diccionarios

Enunciado:
Verifica qué claves se repiten en una lista de diccionarios.

```
lista = [{'a': 1, 'b': 2}, {'b': 3, 'c': 4}, {'a': 5, 'd': 6}]
```

Explicación:
Llevamos un conteo de cuántas veces aparece cada clave.

Solución:

```
from collections import Counter
claves = Counter(k for d in lista for k in d)
```

```
duplicadas = [k for k, v in claves.items() if v > 1]
print(duplicadas)
```

---

9. Convertir diccionario anidado en lista plana

Enunciado:
Convierte un diccionario de diccionarios en una lista de tuplas (clave1, clave2, valor).

```
d = {'A': {'x': 1, 'y': 2}, 'B': {'z': 3}}
```

Explicación:
Iteramos sobre los niveles del diccionario.

Solución:

```
plano = [(k1, k2, v) for k1, sub in d.items() for k2, v in
sub.items()]
print(plano)
```

---

10. Detectar valores duplicados

Enunciado:
Verifica si hay valores duplicados en un diccionario.

```
d = {'a': 1, 'b': 2, 'c': 1}
```

Explicación:
Comparamos la longitud del set con la original.

Solución:

```
duplicados = len(set(d.values())) < len(d)
print(duplicados)
```

---

11. Diccionario de frecuencias de caracteres

Enunciado:
Crea un diccionario con la frecuencia de cada carácter en una cadena.

```
texto = "abracadabra"
```

Explicación:
Contamos con `Counter`.

Solución:

```
from collections import Counter
frecuencias = Counter(texto)
```

```
print(frecuencias)
```

---

12. Crear un índice invertido de palabras por letra inicial

Enunciado:
Agrupa palabras por su letra inicial.

```
palabras = ['barco', 'avión', 'auto', 'bici', 'bar']
```

Explicación:
Usamos la primera letra como clave.

Solución:

```
from collections import defaultdict
indice = defaultdict(list)
for p in palabras:
    indice[p[0]].append(p)
print(dict(indice))
```

---

13. Detectar claves con valores `None`

Enunciado:
Muestra las claves con valor `None`.

```
d = {'a': 1, 'b': None, 'c': 3, 'd': None}
```

Explicación:
Filtramos por el valor.

Solución:

```
nulas = [k for k, v in d.items() if v is None]
print(nulas)
```

---

14. Actualizar diccionario solo si clave no existe

Enunciado:
Agrega una clave solo si no está ya en el diccionario.

```
d = {'a': 1}
```

Explicación:
Usamos `setdefault`.

Solución:

```
d.setdefault('b', 2)
print(d)
```

---

15. Intersección de dos diccionarios por claves

Enunciado:
Crear un nuevo diccionario solo con las claves comunes a dos diccionarios.

```
d1 = {'a': 1, 'b': 2, 'c': 3}
d2 = {'b': 4, 'c': 5, 'd': 6}
```

Explicación:
Usamos intersección de sets de claves.

Solución:

```
interseccion = {k: d1[k] for k in d1.keys() & d2.keys()}
print(interseccion)
```

---

16. Diccionario a lista ordenada de tuplas

Enunciado:
Convierte un diccionario en lista de tuplas ordenadas por valor.

```
d = {'a': 3, 'b': 1, 'c': 2}
```

Explicación:
Ordenamos `items()` con `sorted`.

Solución:

```
lista_ordenada = sorted(d.items(), key=lambda x: x[1])
print(lista_ordenada)
```

---

17. Diccionario de listas a lista de diccionarios

Enunciado:
Transforma este diccionario:

```
d = {'nombre': ['Ana', 'Luis'], 'edad': [23, 30]}
```

en una lista:

```
[{'nombre': 'Ana', 'edad': 23}, {'nombre': 'Luis', 'edad': 30}]
```

Explicación:
Iteramos con `zip`.

Solución:

```
resultado = [dict(zip(d, valores)) for valores in
zip(*d.values())]
print(resultado)
```

---

18. Reemplazar valores en un diccionario según un mapa

Enunciado:
Usa un mapa para traducir los valores de un diccionario.

```
colores = {'a': 'rojo', 'b': 'azul'}
mapa = {'rojo': '#FF0000', 'azul': '#0000FF'}
```

Explicación:
Recorremos y reemplazamos por el valor mapeado.

Solución:

```
traducido = {k: mapa[v] for k, v in colores.items()}
print(traducido)
```

---

19. Contar número de claves únicas en varios diccionarios

Enunciado:
Cuenta cuántas claves distintas hay en total.

```
d1 = {'a': 1, 'b': 2}
d2 = {'b': 3, 'c': 4}
```

Explicación:
Unimos los conjuntos de claves.

Solución:

```
total = len(set(d1) | set(d2))
print(total)
```

---

20. Crear diccionario de posiciones de palabras

Enunciado:
Dada una lista de palabras, crea un diccionario que indique sus posiciones en la lista.

```
palabras = ['sol', 'luna', 'mar', 'luna', 'sol']
```

Explicación:
Recorremos con `enumerate` y almacenamos en listas.

Solución:

```
from collections import defaultdict
posiciones = defaultdict(list)
for i, p in enumerate(palabras):
    posiciones[p].append(i)
print(dict(posiciones))
```

---

8. Funciones

Dificultad facil

1. Función que sume dos números

Enunciado: Crea una función `suma(a, b)` que devuelva la suma de dos números.

Explicación: Solo hay que retornar el resultado de `a + b`.

Solución:

```
def suma(a, b):
    return a + b

print(suma(3, 5))  # 8
```

---

2. Función que devuelva el cuadrado de un número

Enunciado: Crea una función `cuadrado(n)` que devuelva el cuadrado de un número.

Explicación: Elevar un número al cuadrado es multiplicarlo por sí mismo.

Solución:

```
def cuadrado(n):
    return n * n

print(cuadrado(4))  # 16
```

---

3. Función que diga si un número es par

Enunciado: Crea una función `es_par(n)` que devuelva `True` si el número es par y `False` si no.

Explicación: Un número es par si al dividirlo entre 2 el resto es 0.

Solución:

```
def es_par(n):
    return n % 2 == 0

print(es_par(6))  # True
```

---

4. Función que devuelva el mayor de dos números

Enunciado: Crea una función `mayor(a, b)` que devuelva el mayor de los dos números.

Explicación: Compara `a` y `b` y retorna el más grande.

Solución:

```
def mayor(a, b):
    return a if a > b else b

print(mayor(10, 20))  # 20
```

---

5. Función que concatene dos cadenas

Enunciado: Crea una función `concatena(a, b)` que devuelva las dos cadenas unidas.

Explicación: Usamos el operador `+` para unir cadenas.

Solución:

```
def concatena(a, b):
    return a + b

print(concatena("Hola ", "mundo"))  # Hola mundo
```

---

6. Función que calcule el área de un triángulo

Enunciado: Crea una función `area_triangulo(base, altura)`.

Explicación: Fórmula: `(base * altura) / 2`.

Solución:

```
def area_triangulo(base, altura):
    return (base * altura) / 2

print(area_triangulo(10, 5))  # 25.0
```

---

7. Función que cuente las letras de una cadena

Enunciado: Crea una función `contar_letras(cadena)`.

Explicación: Usa `len()` para contar caracteres.

Solución:

```
def contar_letras(cadena):
    return len(cadena)

print(contar_letras("python"))  # 6
```

---

8. Función que invierta una cadena

Enunciado: Crea una función `invertir(cadena)`.

Explicación: Usa slicing con paso `-1`.

Solución:

```
def invertir(cadena):
    return cadena[::-1]

print(invertir("hola"))  # aloh
```

---

9. Función que cuente cuántas vocales hay

Enunciado: Crea una función `contar_vocales(cadena)`.

Explicación: Recorre la cadena contando vocales.

Solución:

```
def contar_vocales(cadena):
    vocales = "aeiouAEIOU"
```

```
    return sum(1 for letra in cadena if letra in vocales)

print(contar_vocales("Hola mundo"))  # 4
```

---

10. Función que calcule el factorial

Enunciado: Crea una función `factorial(n)`.

Explicación: Multiplica todos los enteros desde 1 hasta `n`.

Solución:

```
def factorial(n):
    resultado = 1
    for i in range(1, n+1):
        resultado *= i
    return resultado

print(factorial(5))  # 120
```

---

11. Función que devuelva si un número es positivo, negativo o cero

Enunciado: Crea `analizar_numero(n)` que devuelva un texto según el valor.

Explicación: Usa condicionales `if`.

Solución:

```
def analizar_numero(n):
    if n > 0:
        return "Positivo"
    elif n < 0:
        return "Negativo"
    else:
        return "Cero"

print(analizar_numero(-3))  # Negativo
```

---

12. Función que devuelva la suma de una lista

Enunciado: Crea `suma_lista(lista)`.

Explicación: Usa un bucle o la función `sum()`.

Solución:

```
def suma_lista(lista):
    return sum(lista)

print(suma_lista([1, 2, 3]))  # 6
```

---

13. Función que devuelva el número más pequeño en una lista

Enunciado: Crea `minimo(lista)`.

Explicación: Usa `min()`.

Solución:

```
def minimo(lista):
    return min(lista)

print(minimo([4, 1, 9]))  # 1
```

---

14. Función que verifique si un carácter es vocal

Enunciado: Crea `es_vocal(letra)`.

Explicación: Compara contra las vocales.

Solución:

```
def es_vocal(letra):
    return letra.lower() in "aeiou"

print(es_vocal("E"))  # True
```

---

15. Función que cuente cuántos elementos tiene una lista

Enunciado: Crea `contar_elementos(lista)`.

Explicación: Usa `len()`.

Solución:

```
def contar_elementos(lista):
    return len(lista)

print(contar_elementos([1, 2, 3, 4]))  # 4
```

---

16. Función que calcule la media de una lista

Enunciado: Crea `media(lista)`.

Explicación: Suma total entre número de elementos.

Solución:

```
def media(lista):
    return sum(lista) / len(lista)

print(media([5, 10, 15]))  # 10.0
```

---

17. Función que convierta Celsius a Fahrenheit

Enunciado: Crea `celsius_a_fahrenheit(c)`.

Explicación: Fórmula: `F = C * 9/5 + 32`.

Solución:

```
def celsius_a_fahrenheit(c):
    return c * 9/5 + 32

print(celsius_a_fahrenheit(0))  # 32.0
```

---

18. Función que diga si una palabra es palíndromo

Enunciado: Crea `es_palindromo(palabra)`.

Explicación: Compara la palabra con su reverso.

Solución:

```
def es_palindromo(palabra):
    return palabra == palabra[::-1]
```

```
print(es_palindromo("oso"))  # True
```

---

19. Función que multiplique todos los elementos de una lista

Enunciado: Crea `multiplicar_lista(lista)`.

Explicación: Usa un bucle y acumula el producto.

Solución:

```
def multiplicar_lista(lista):
    resultado = 1
    for num in lista:
        resultado *= num
    return resultado

print(multiplicar_lista([2, 3, 4]))  # 24
```

---

20. Función que devuelva la longitud de la palabra más larga

Enunciado: Crea `longitud_max(palabras)`.

Explicación: Recorre la lista y encuentra la mayor longitud.

Solución:

```
def longitud_max(palabras):
    return max(len(p) for p in palabras)

print(longitud_max(["hola", "murciélago", "sol"]))  # 10
```

Dificultad media

---

1. Suma de una lista

Enunciado:
Crea una función `suma_lista(lista)` que reciba una lista de números y devuelva la suma de todos los elementos.

Explicación:
Usamos un bucle `for` para recorrer la lista y acumular la suma.

Solución:

```
def suma_lista(lista):
    total = 0
    for num in lista:
        total += num
    return total

# Ejemplo
print(suma_lista([1, 2, 3, 4]))  # 10
```

---

2. Número primo

Enunciado:
Crea una función `es_primo(n)` que devuelva `True` si `n` es primo, y `False` si no lo es.

Explicación:
Probamos si `n` tiene divisores entre 2 y la raíz cuadrada de `n`.

Solución:

```
def es_primo(n):
    if n < 2:
        return False
    for i in range(2, int(n**0.5)+1):
        if n % i == 0:
            return False
    return True

print(es_primo(7))  # True
```

---

3. Contar vocales

Enunciado:
Escribe una función `contar_vocales(cadena)` que cuente cuántas vocales hay en una cadena.

Explicación:
Comparamos cada carácter con un conjunto de vocales.

Solución:

```
def contar_vocales(cadena):
    vocales = 'aeiouAEIOU'
    return sum(1 for letra in cadena if letra in vocales)

print(contar_vocales("Hola Mundo"))  # 4
```

---

4. Factorial

Enunciado:
Crea una función `factorial(n)` que devuelva el factorial de un número entero positivo.

Explicación:
Usamos un bucle `for` para multiplicar del 1 a `n`.

Solución:

```
def factorial(n):
    resultado = 1
    for i in range(2, n + 1):
        resultado *= i
    return resultado

print(factorial(5))  # 120
```

---

5. Palíndromo

Enunciado:
Escribe una función `es_palindromo(palabra)` que diga si una palabra se lee igual al derecho que al revés.

Explicación:
Comparamos la cadena original con su reverso.

Solución:

```
def es_palindromo(palabra):
    return palabra == palabra[::-1]

print(es_palindromo("radar"))  # True
```

---

6. Mayor de tres

Enunciado:
Define una función `mayor_de_tres(a, b, c)` que retorne el mayor de los tres números.

Explicación:
Usamos `max()` o condicionales `if`.

Solución:

```
def mayor_de_tres(a, b, c):
    return max(a, b, c)

print(mayor_de_tres(3, 9, 5))  # 9
```

---

7. Conversión Celsius a Fahrenheit

Enunciado:
Crea una función `celsius_a_fahrenheit(c)` que convierta grados Celsius a Fahrenheit.

Explicación:
Usamos la fórmula: F = C * 9/5 + 32.

Solución:

```
def celsius_a_fahrenheit(c):
    return c * 9 / 5 + 32

print(celsius_a_fahrenheit(0))  # 32.0
```

---

8. Número perfecto

Enunciado:
Crea una función `es_perfecto(n)` que determine si un número es perfecto.

Explicación:
La suma de sus divisores (excluyendo el número) debe ser igual a él.

Solución:

```
def es_perfecto(n):
    suma = sum(i for i in range(1, n) if n % i == 0)
    return suma == n

print(es_perfecto(28))  # True
```

---

9. Intercambiar valores

Enunciado:
Haz una función `intercambiar(a, b)` que devuelva los dos valores intercambiados.

Explicación:
Intercambiamos usando múltiples asignaciones.

Solución:

```
def intercambiar(a, b):
    return b, a

print(intercambiar(3, 5))  # (5, 3)
```

---

10. Contar palabras

Enunciado:
Escribe una función `contar_palabras(frase)` que cuente cuántas palabras tiene una frase.

Explicación:
Usamos `split()` para separar las palabras.

Solución:

```
def contar_palabras(frase):
    return len(frase.split())

print(contar_palabras("Hola mundo feliz"))  # 3
```

---

11. Promedio de una lista

Enunciado:
Escribe una función `promedio(lista)` que calcule el promedio de los valores.

Explicación:
Sumamos los valores y dividimos entre la longitud.

Solución:

```
def promedio(lista):
    return sum(lista) / len(lista) if lista else 0


print(promedio([2, 4, 6]))  # 4.0
```

---

12. Números pares en una lista

Enunciado:
Crea una función `pares(lista)` que devuelva una lista con los pares.

Explicación:
Usamos comprensión de listas con `if`.

Solución:

```
def pares(lista):
    return [n for n in lista if n % 2 == 0]


print(pares([1, 2, 3, 4]))  # [2, 4]
```

---

13. Potencia sin usar `**`

Enunciado:
Crea una función `potencia(base, exponente)` sin usar el operador `**`.

Explicación:
Multiplicamos la base consigo misma `exponente` veces.

Solución:

```
def potencia(base, exponente):
    resultado = 1
    for _ in range(exponente):
        resultado *= base
```

```
    return resultado

print(potencia(2, 3))  # 8
```

---

14. Contar dígitos

Enunciado:
Crea una función `contar_digitos(n)` que cuente cuántos dígitos tiene un número.

Explicación:
Convertimos a cadena y usamos `len()`.

Solución:

```
def contar_digitos(n):
    return len(str(abs(n)))

print(contar_digitos(12345))  # 5
```

---

15. Invertir lista

Enunciado:
Crea una función `invertir_lista(lista)` que devuelva una lista invertida.

Explicación:
Usamos slicing `[::-1]`.

Solución:

```
def invertir_lista(lista):
    return lista[::-1]

print(invertir_lista([1, 2, 3]))  # [3, 2, 1]
```

---

16. Buscar mayor impar

Enunciado:
Crea una función `mayor_impar(lista)` que devuelva el mayor número impar.

Explicación:
Filtramos los impares y usamos `max()`.

Solución:

```
def mayor_impar(lista):
    impares = [n for n in lista if n % 2 != 0]
    return max(impares) if impares else None

print(mayor_impar([2, 3, 8, 7]))  # 7
```

---

17. Números únicos

Enunciado:
Escribe una función `numeros_unicos(lista)` que devuelva los números sin repetir.

Explicación:
Convertimos la lista en conjunto `set()`.

Solución:

```
def numeros_unicos(lista):
    return list(set(lista))

print(numeros_unicos([1, 2, 2, 3, 1]))  # [1, 2, 3]
```

---

18. Producto de una lista

Enunciado:
Crea una función `producto_lista(lista)` que devuelva el producto de los números.

Explicación:
Multiplicamos todos los elementos.

Solución:

```
def producto_lista(lista):
    resultado = 1
    for n in lista:
        resultado *= n
    return resultado

print(producto_lista([2, 3, 4]))  # 24
```

---

19. Sumar dígitos

Enunciado:
Crea una función `suma_digitos(n)` que devuelva la suma de los dígitos de un número.

Explicación:
Convertimos a cadena y sumamos dígito por dígito.

Solución:

```
def suma_digitos(n):
    return sum(int(d) for d in str(abs(n)))

print(suma_digitos(123))  # 6
```

---

20. Verificar anagrama

Enunciado:
Escribe una función `es_anagrama(p1, p2)` que diga si dos palabras son anagramas.

Explicación:
Un anagrama tiene las mismas letras en diferente orden.

Solución:

```
def es_anagrama(p1, p2):
    return sorted(p1.lower()) == sorted(p2.lower())

print(es_anagrama("roma", "amor"))  # True
```

Dificultad avanzada

1. Generador de funciones derivadas numéricas

Enunciado: Crea una función que reciba otra función `f(x)` y un valor `x`, y devuelva una estimación de la derivada en `x` usando la fórmula del límite.

Explicación: Usamos el concepto de derivada como el límite del cociente de incrementos.

Solución:

```
def derivada(f, x, h=1e-5):
    return (f(x + h) - f(x - h)) / (2 * h)
```

```
# Ejemplo
print(derivada(lambda x: x**2, 3))  # ≈ 6
```

---

2. Memoización manual de Fibonacci

Enunciado: Implementa una función de Fibonacci que recuerde los valores ya calculados (memoización sin usar `lru_cache`).

Explicación: Guardamos los resultados en un diccionario para evitar cálculos repetidos.

Solución:

```
def fib(n, memo={}):
    if n in memo: return memo[n]
    if n <= 2: return 1
    memo[n] = fib(n-1, memo) + fib(n-2, memo)
    return memo[n]

print(fib(35))
```

---

3. Aplicación de múltiples funciones

Enunciado: Crea una función `aplica_todo(x, funcs)` que reciba un número y una lista de funciones, y devuelva una lista con los resultados.

Explicación: Se recorre la lista de funciones y se aplican todas al valor dado.

Solución:

```
def aplica_todo(x, funcs):
    return [f(x) for f in funcs]

print(aplica_todo(2, [lambda x: x+1, lambda x: x**2, lambda x: 1/x]))
```

---

4. Generador de funciones polinómicas

Enunciado: Escribe una función que genere otra función polinómica dados los coeficientes.

Explicación: Se crea una función lambda que representa un polinomio de grado n.

Solución:

```
def genera_polinomio(coef):
    return lambda x: sum(c * x**i for i, c in enumerate(coef))

f = genera_polinomio([2, 3, 0, 1])  # 2 + 3x + x³
print(f(2))  # 2 + 6 + 8 = 16
```

---

5. Función inversa numérica

Enunciado: Implementa una función que calcule la inversa numérica de una función `f(x)` usando búsqueda binaria.

Explicación: Suponiendo que `f` es monótona, se puede invertir iterativamente.

Solución:

```
def inversa(f, y, tol=1e-5):
    a, b = 0, max(1, y)
    while b - a > tol:
        m = (a + b) / 2
        if f(m) < y:
            a = m
        else:
            b = m
    return (a + b) / 2

print(inversa(lambda x: x**2, 16))  # ≈ 4
```

---

6. Decorador de cronómetro

Enunciado: Crea un decorador que imprima el tiempo que tarda en ejecutarse una función.

Explicación: Usamos el módulo `time` para medir la duración de ejecución.

Solución:

```
import time

def cronometro(f):
    def wrapper(*args, **kwargs):
```

```
        inicio = time.time()
        res = f(*args, **kwargs)
        print("Duración:", time.time() - inicio, "s")
        return res
    return wrapper

@cronometro
def prueba():
    sum(i for i in range(10**6))

prueba()
```

---

7. Validación de argumentos con decorador

Enunciado: Crea un decorador que valide que todos los argumentos sean positivos.

Explicación: Se verifican los argumentos antes de ejecutar la función original.

Solución:

```
def positivos(f):
    def wrapper(*args):
        if any(x <= 0 for x in args):
            raise ValueError("Todos los argumentos deben ser
positivos")
        return f(*args)
    return wrapper

@positivos
def suma(a, b):
    return a + b

print(suma(3, 4))
```

---

8. Función de orden superior para integración numérica

Enunciado: Crea una función que integre otra función usando el método del trapecio.

Explicación: Se suman áreas de pequeños trapecios formados bajo la curva.

Solución:

```
def integrar(f, a, b, n=1000):
    h = (b - a) / n
    return (f(a) + f(b) + 2*sum(f(a + i*h) for i in range(1, n))) * h / 2

print(integrar(lambda x: x**2, 0, 3))  # ≈ 9
```

---

9. Parámetros arbitrarios de tipo `**kwargs`

Enunciado: Escribe una función que reciba parámetros arbitrarios y devuelva una cadena con claves y valores.

Explicación: Se formatea el diccionario para mostrar su contenido.

Solución:

```
def mostrar_kwargs(**kwargs):
    return ", ".join(f"{k}={v}" for k, v in kwargs.items())

print(mostrar_kwargs(nombre="Ana", edad=30))
```

---

10. Función curryficada

Enunciado: Implementa una función `curry_suma` que reciba un número y devuelva otra función que sume con ese número.

Explicación: Se usa cierre de funciones (closures).

Solución:

```
def curry_suma(n):
    return lambda x: x + n

suma5 = curry_suma(5)
print(suma5(3))  # 8
```

---

11. Aplicación recursiva sobre listas anidadas

Enunciado: Escribe una función que sume todos los enteros dentro de una lista anidada.

Explicación: Se usa recursión para recorrer listas dentro de listas.

Solución:

```
def suma_anidada(lista):
    total = 0
    for x in lista:
        if isinstance(x, list):
            total += suma_anidada(x)
        else:
            total += x
    return total

print(suma_anidada([1, [2, [3, 4]], 5]))  # 15
```

---

12. Decorador de logging

Enunciado: Crea un decorador que imprima el nombre de la función y sus argumentos antes de ejecutarla.

Explicación: Se intercepta la llamada y se imprime la información.

Solución:

```
def logger(f):
    def wrapper(*args, **kwargs):
        print(f"Llamando {f.__name__} con args={args},
kwargs={kwargs}")
        return f(*args, **kwargs)
    return wrapper

@logger
def resta(a, b):
    return a - b
```

```
print(resta(10, 3))
```

---

13. Conversión de base numérica personalizada

Enunciado: Crea una función que convierta un número entero a cualquier base entre 2 y 36.

Explicación: Se divide el número y se usan caracteres alfanuméricos.

Solución:

```
def a_base(n, base):
    digitos = "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ"
    if n < base:
        return digitos[n]
    else:
        return a_base(n // base, base) + digitos[n % base]

print(a_base(255, 16))  # FF
```

---

14. Verificación de palíndromos recursivos

Enunciado: Escribe una función recursiva que verifique si una cadena es palíndroma.

Explicación: Se comparan los extremos de la cadena recursivamente.

Solución:

```
def es_palindromo(s):
    if len(s) <= 1:
        return True
    if s[0] != s[-1]:
        return False
    return es_palindromo(s[1:-1])

print(es_palindromo("reconocer"))  # True
```

---

15. Conteo de palabras únicas con función

Enunciado: Escribe una función que devuelva un diccionario con las frecuencias de cada palabra en un texto.

Explicación: Se usa un diccionario para contar ocurrencias.

Solución:

```
def contar_palabras(texto):
    palabras = texto.lower().split()
    frec = {}
    for palabra in palabras:
        frec[palabra] = frec.get(palabra, 0) + 1
    return frec

print(contar_palabras("hola mundo hola mundo mundo"))
```

---

16. Simulación de funciones lambda anidadas

Enunciado: Crea una función que devuelva una función anidada de la forma `f(x)(y)(z) = x + y * z`.

Explicación: Se utilizan funciones lambda anidadas para construir una función en múltiples pasos.

Solución:

```
def anidada():
    return lambda x: lambda y: lambda z: x + y * z

f = anidada()
print(f(2)(3)(4))  # 2 + 3*4 = 14
```

---

17. Función generadora de permutaciones

Enunciado: Implementa una función que genere todas las permutaciones posibles de una lista sin usar `itertools`.

Explicación: Se usa recursión para elegir cada elemento como inicial y permutar el resto.

Solución:

```
def permutaciones(lista):
    if len(lista) == 0:
        return [[]]
    res = []
```

```
    for i in range(len(lista)):
        resto = lista[:i] + lista[i+1:]
        for p in permutaciones(resto):
            res.append([lista[i]] + p)
    return res

print(permutaciones([1, 2, 3]))
```

---

18. Compresor de cadenas simple

Enunciado: Escribe una función que comprima cadenas reemplazando secuencias repetidas por carácter + número de repeticiones: `"aaabb"` → `"a3b2"`.

Explicación: Se recorre la cadena contando repeticiones consecutivas.

Solución:

```
def comprimir(cadena):
    if not cadena:
        return ""
    res = ""
    actual = cadena[0]
    contador = 1
    for c in cadena[1:]:
        if c == actual:
            contador += 1
        else:
            res += actual + str(contador)
            actual = c
            contador = 1
    res += actual + str(contador)
    return res

print(comprimir("aaabbcccc"))  # a3b2c4
```

---

19. Evaluador de expresiones matemáticas simples

Enunciado: Escribe una función que evalúe una expresión matemática en formato cadena (`"3 + 4 * 2"`) sin usar `eval`.

Explicación: Se hace parsing de la cadena y se aplica la precedencia de operadores.

Solución:

```
def evaluar(expr):
    import operator
    ops = {'+': operator.add, '-': operator.sub, '*': operator.mul, '/': operator.truediv}
    def parse(tokens):
        stack = []
        num = ""
        for tok in tokens:
            if tok in ops:
                stack.append(float(num))
                stack.append(tok)
                num = ""
            else:
                num += tok
        stack.append(float(num))

        while '*' in stack or '/' in stack:
            for i in range(len(stack)):
                if stack[i] in '*/':
                    op = ops[stack[i]]
                    res = op(stack[i-1], stack[i+1])
                    stack = stack[:i-1] + [res] + stack[i+2:]
                    break

        while '+' in stack or '-' in stack:
            for i in range(len(stack)):
```

```
                if stack[i] in '+-':
                    op = ops[stack[i]]
                    res = op(stack[i-1], stack[i+1])
                    stack = stack[:i-1] + [res] + stack[i+2:]
                    break
        return stack[0]

    return parse(expr.replace(" ", ""))

print(evaluar("3 + 4 * 2"))  # 11
```

---

20. Decorador con parámetros

Enunciado: Escribe un decorador que acepte un mensaje como parámetro y lo imprima antes de ejecutar la función.

Explicación: Requiere crear un decorador que a su vez devuelve otro decorador.

Solución:

```
def con_mensaje(mensaje):
    def decorador(f):
        def wrapper(*args, **kwargs):
            print(mensaje)
            return f(*args, **kwargs)
        return wrapper
    return decorador

@con_mensaje("Ejecutando la función...")
def sumar(a, b):
    return a + b

print(sumar(3, 5))
```

---

9. Programación orientada a objetos
Dificultad facil

---

1. Crear una clase `Perro`

Enunciado:
Crea una clase llamada `Perro` con atributos `nombre` y `edad`. Incluye un método `ladrar` que imprima “¡Guau!”.

Explicación:
Este ejercicio enseña a crear una clase con atributos y un método simple.

Solución:

```
class Perro:
    def __init__(self, nombre, edad):
        self.nombre = nombre
        self.edad = edad

    def ladrar(self):
        print("¡Guau!")

p = Perro("Firulais", 3)
p.ladrar()
```

---

2. Clase `Coche` con método `arrancar`

Enunciado:
Crea una clase `Coche` con un atributo `marca` y un método `arrancar` que imprima “El coche está en marcha”.

Explicación:
Este ejercicio refuerza el concepto de métodos de instancia.

Solución:

```
class Coche:
    def __init__(self, marca):
        self.marca = marca
```

```
    def arrancar(self):
        print("El coche está en marcha")

c = Coche("Toyota")
c.arrancar()
```

---

3. Clase `Rectángulo` con cálculo de área

Enunciado:
Crea una clase `Rectangulo` con atributos `base` y `altura`, y un método `area` que devuelva su área.

Explicación:
Se practica el uso de métodos que devuelven valores.

Solución:

```
class Rectangulo:
    def __init__(self, base, altura):
        self.base = base
        self.altura = altura

    def area(self):
        return self.base * self.altura

r = Rectangulo(5, 3)
print(r.area())  # 15
```

---

4. Clase `Libro` con título y autor

Enunciado:
Define una clase `Libro` con atributos `titulo` y `autor`, y un método que imprima esa información.

Explicación:
Ejercicio para representar objetos con múltiples atributos.

Solución:

```
class Libro:
```

```
    def __init__(self, titulo, autor):
        self.titulo = titulo
        self.autor = autor

    def mostrar_info(self):
        print(f"{self.titulo} por {self.autor}")

libro = Libro("1984", "George Orwell")
libro.mostrar_info()
```

---

5. Clase `CuentaBancaria` con saldo

Enunciado:
Crea una clase `CuentaBancaria` con atributo `saldo` y métodos `depositar` y `retirar`.

Explicación:
Introduce modificación de atributos.

Solución:

```
class CuentaBancaria:
    def __init__(self, saldo=0):
        self.saldo = saldo

    def depositar(self, cantidad):
        self.saldo += cantidad

    def retirar(self, cantidad):
        if cantidad <= self.saldo:
            self.saldo -= cantidad

cuenta = CuentaBancaria()
cuenta.depositar(100)
cuenta.retirar(40)
print(cuenta.saldo)  # 60
```

---

6. Clase `Estudiante` con promedio

Enunciado:
Define una clase `Estudiante` con notas y un método que calcule el promedio.

Explicación:
Trabaja con listas y lógica básica dentro de una clase.

Solución:

```
class Estudiante:
    def __init__(self, nombre, notas):
        self.nombre = nombre
        self.notas = notas

    def promedio(self):
        return sum(self.notas) / len(self.notas)

e = Estudiante("Ana", [8, 7, 9])
print(e.promedio())  # 8.0
```

---

7. Clase `Persona` con saludo personalizado

Enunciado:
Crea una clase `Persona` que salude usando su nombre.

Explicación:
Uso de strings y atributos básicos.

Solución:

```
class Persona:
    def __init__(self, nombre):
        self.nombre = nombre

    def saludar(self):
        print(f"Hola, soy {self.nombre}")

p = Persona("Lucía")
```

```
p.saludar()
```

---

8. Clase `Producto` con descuento

Enunciado:
Crea una clase `Producto` con nombre y precio, y un método para aplicar un descuento.

Explicación:
Se introduce una operación matemática básica.

Solución:

```
class Producto:
    def __init__(self, nombre, precio):
        self.nombre = nombre
        self.precio = precio

    def aplicar_descuento(self, porcentaje):
        self.precio -= self.precio * porcentaje / 100

p = Producto("Camiseta", 20)
p.aplicar_descuento(10)
print(p.precio)  # 18.0
```

---

9. Clase `Circulo` con cálculo de área

Enunciado:
Crea una clase `Circulo` con radio y un método `area`.

Explicación:
Introduce el uso de la constante `pi`.

Solución:

```
import math

class Circulo:
    def __init__(self, radio):
        self.radio = radio
```

```
    def area(self):
        return math.pi * self.radio**2

c = Circulo(3)
print(round(c.area(), 2))  # 28.27
```

---

10. Clase `Animal` con herencia

Enunciado:
Crea una clase `Animal` y una clase `Gato` que herede de `Animal`.

Explicación:
Introduce la herencia simple.

Solución:

```
class Animal:
    def hablar(self):
        print("Sonido genérico")

class Gato(Animal):
    def hablar(self):
        print("Miau")

g = Gato()
g.hablar()
```

---

11. Clase `Punto` en el plano

Enunciado:
Crea una clase `Punto` con coordenadas `x` e `y` y un método para mover el punto.

Explicación:
Trabaja con modificación de atributos.

Solución:

```
class Punto:
    def __init__(self, x, y):
        self.x = x
```

```
        self.y = y

    def mover(self, dx, dy):
        self.x += dx
        self.y += dy

p = Punto(1, 2)
p.mover(3, -1)
print(p.x, p.y)  # 4, 1
```

---

12. Clase `Cancion` con duración

Enunciado:
Crea una clase `Cancion` con atributos `titulo` y `duracion`. Imprime la duración en minutos y segundos.

Explicación:
Practica el manejo de enteros y formato de tiempo.

Solución:

```
class Cancion:
    def __init__(self, titulo, duracion):
        self.titulo = titulo
        self.duracion = duracion  # en segundos

    def mostrar_duracion(self):
        minutos = self.duracion // 60
        segundos = self.duracion % 60
        print(f"{minutos}m {segundos}s")

c = Cancion("Imagine", 185)
c.mostrar_duracion()
```

---

13. Clase `Usuario` con contraseña

Enunciado:
Crea una clase `Usuario` que almacene nombre y contraseña. Verifica si una contraseña dada es correcta.

Explicación:
Introduce lógica condicional.

Solución:

```
class Usuario:
    def __init__(self, nombre, contraseña):
        self.nombre = nombre
        self.contraseña = contraseña

    def verificar(self, contraseña_intento):
        return self.contraseña == contraseña_intento

u = Usuario("admin", "1234")
print(u.verificar("1234"))  # True
```

---

14. Clase `Contador` que se incrementa

Enunciado:
Crea una clase `Contador` con un método para incrementar su valor.

Explicación:
Introduce el uso de métodos para modificar atributos internos.

Solución:

```
class Contador:
    def __init__(self):
        self.valor = 0

    def incrementar(self):
        self.valor += 1

c = Contador()
```

```
c.incrementar()
c.incrementar()
print(c.valor)  # 2
```

---

15. Clase `Empleado` con sueldo y bonificación

Enunciado:
Crea una clase `Empleado` con un método para aplicar una bonificación del 10%.

Explicación:
Se practica la modificación de atributos numéricos.

Solución:

```
class Empleado:
    def __init__(self, nombre, sueldo):
        self.nombre = nombre
        self.sueldo = sueldo

    def bonificar(self):
        self.sueldo *= 1.10

e = Empleado("Carlos", 1000)
e.bonificar()
print(e.sueldo)  # 1100.0
```

---

16. Clase `Reloj` que avanza el tiempo

Enunciado:
Crea una clase `Reloj` que avance el tiempo en horas y minutos.

Explicación:
Refuerza lógica de formato de reloj.

Solución:

```
class Reloj:
    def __init__(self, horas, minutos):
        self.horas = horas
        self.minutos = minutos
```

```
    def avanzar(self, mins):
        self.minutos += mins
        self.horas += self.minutos // 60
        self.minutos %= 60
        self.horas %= 24

r = Reloj(22, 50)
r.avanzar(80)
print(r.horas, r.minutos)  # 0, 10
```

---

17. Clase `Temperatura` con conversión

Enunciado:
Crea una clase `Temperatura` que convierta de Celsius a Fahrenheit.

Explicación:
Practica el uso de fórmulas simples en métodos.

Solución:

```
class Temperatura:
    def __init__(self, celsius):
        self.celsius = celsius

    def a_fahrenheit(self):
        return self.celsius * 9 / 5 + 32

t = Temperatura(20)
print(t.a_fahrenheit())  # 68.0
```

---

18. Clase `Persona` con método `es_mayor`

Enunciado:
Agrega un método `es_mayor` a una clase `Persona` que devuelva si la persona es mayor de edad.

Explicación:
Introduce una condición booleana en un método.

Solución:

```
class Persona:
    def __init__(self, nombre, edad):
        self.nombre = nombre
        self.edad = edad

    def es_mayor(self):
        return self.edad >= 18

p = Persona("Eva", 20)
print(p.es_mayor())  # True
```

---

19. Clase `Pelicula` con método `es_larga`

Enunciado:
Define una clase `Pelicula` que devuelva si dura más de 2 horas.

Explicación:
Uso de condiciones en clases reales.

Solución:

```
class Pelicula:
    def __init__(self, titulo, duracion):
        self.titulo = titulo
        self.duracion = duracion

    def es_larga(self):
        return self.duracion > 120

p = Pelicula("Titanic", 195)
print(p.es_larga())  # True
```

---

20. Clase `Cuenta` con impresión de estado

Enunciado:
Crea una clase `Cuenta` que imprima su estado actual en una línea.

Explicación:
Utiliza `__str__` para representar el objeto.

Solución:

```
class Cuenta:
    def __init__(self, titular, saldo):
        self.titular = titular
        self.saldo = saldo

    def __str__(self):
        return f"{self.titular}: ${self.saldo}"

c = Cuenta("Laura", 500)
print(c)  # Laura: $500
```

Dificultad media

---

1. Clase Rectángulo

Enunciado: Crea una clase `Rectangulo` con atributos `base` y `altura`. Incluye un método para calcular el área.

Explicación: La clase contiene dos atributos y un método que multiplica ambos.

Solución:

```
class Rectangulo:
    def __init__(self, base, altura):
        self.base = base
        self.altura = altura
```

```
    def area(self):
        return self.base * self.altura

r = Rectangulo(5, 3)
print(r.area())  # 15
```

---

2. Clase Círculo con perímetro

Enunciado: Crea una clase `Circulo` con radio y un método que devuelva el perímetro.

Explicación: Usa la fórmula del perímetro: `2 * pi * radio`.

Solución:

```
import math

class Circulo:
    def __init__(self, radio):
        self.radio = radio

    def perimetro(self):
        return 2 * math.pi * self.radio

c = Circulo(4)
print(round(c.perimetro(), 2))  # 25.13
```

---

3. Contador con incremento

Enunciado: Crea una clase `Contador` que tenga un método `incrementar` y otro `valor`.

Explicación: Se lleva un contador que se incrementa cada vez que se llama al método.

Solución:

```
class Contador:
    def __init__(self):
        self.cuenta = 0

    def incrementar(self):
```

```
        self.cuenta += 1

    def valor(self):
        return self.cuenta

c = Contador()
c.incrementar()
c.incrementar()
print(c.valor())  # 2
```

---

4. Empleado con aumento de sueldo

Enunciado: Crea una clase `Empleado` que tenga nombre, sueldo y un método que aplique un aumento del 10%.

Explicación: Se utiliza una función que multiplica el sueldo por 1.10.

Solución:

```
class Empleado:
    def __init__(self, nombre, sueldo):
        self.nombre = nombre
        self.sueldo = sueldo

    def aumentar_sueldo(self):
        self.sueldo *= 1.10

e = Empleado("Ana", 1000)
e.aumentar_sueldo()
print(e.sueldo)  # 1100.0
```

---

5. Cuenta bancaria

Enunciado: Crea una clase `Cuenta` con métodos para depositar, retirar y mostrar saldo.

Explicación: Se controlan operaciones básicas de una cuenta bancaria.

Solución:

```
class Cuenta:
    def __init__(self, saldo=0):
        self.saldo = saldo

    def depositar(self, monto):
        self.saldo += monto

    def retirar(self, monto):
        if monto <= self.saldo:
            self.saldo -= monto
        else:
            print("Fondos insuficientes")

    def mostrar_saldo(self):
        return self.saldo

c = Cuenta(100)
c.depositar(50)
c.retirar(30)
print(c.mostrar_saldo())  # 120
```

---

6. Clase Estudiante con nota media

Enunciado: Crea una clase `Estudiante` que almacene nombre y una lista de notas. Agrega método `nota_media`.

Explicación: Calcula el promedio de una lista de números.

Solución:

```
class Estudiante:
    def __init__(self, nombre, notas):
        self.nombre = nombre
        self.notas = notas
```

```
    def nota_media(self):
        return sum(self.notas) / len(self.notas)

e = Estudiante("Luis", [7, 8, 9])
print(e.nota_media())  # 8.0
```

---

7. Clase Vehículo con herencia

Enunciado: Crea una clase `Vehiculo` y dos clases hijas `Coche` y `Moto` con un método `tipo`.

Explicación: Se practica la herencia y el polimorfismo.

Solución:

```
class Vehiculo:
    def tipo(self):
        return "Vehículo"

class Coche(Vehiculo):
    def tipo(self):
        return "Coche"

class Moto(Vehiculo):
    def tipo(self):
        return "Moto"

v = Coche()
print(v.tipo())  # Coche
```

---

8. Clase Persona con validación

Enunciado: Crea una clase `Persona` que valide que la edad sea mayor que 0.

Explicación: Validamos el atributo en el constructor.

Solución:

```
class Persona:
    def __init__(self, nombre, edad):
```

```
        if edad <= 0:
            raise ValueError("Edad no válida")
        self.nombre = nombre
        self.edad = edad

p = Persona("Carlos", 30)
print(p.nombre)  # Carlos
```

---

9. Producto con descuento

Enunciado: Crea una clase `Producto` con nombre, precio y un método que aplique descuento.

Explicación: Se calcula el precio tras aplicar el porcentaje.

Solución:

```
class Producto:
    def __init__(self, nombre, precio):
        self.nombre = nombre
        self.precio = precio

    def aplicar_descuento(self, porcentaje):
        self.precio *= (1 - porcentaje / 100)

p = Producto("Zapatos", 100)
p.aplicar_descuento(20)
print(p.precio)  # 80.0
```

---

10. Clase Fracción con suma

Enunciado: Implementa una clase `Fraccion` que permita sumar dos fracciones.

Explicación: Usa la fórmula de suma de fracciones con denominador común.

Solución:

```
from math import gcd

class Fraccion:
```

```
    def __init__(self, num, den):
        self.num = num
        self.den = den

    def sumar(self, otra):
        nuevo_num = self.num * otra.den + otra.num * self.den
        nuevo_den = self.den * otra.den
        comun = gcd(nuevo_num, nuevo_den)
        return Fraccion(nuevo_num // comun, nuevo_den // comun)

    def mostrar(self):
        return f"{self.num}/{self.den}"

f1 = Fraccion(1, 2)
f2 = Fraccion(1, 3)
f3 = f1.sumar(f2)
print(f3.mostrar())  # 5/6
```

---

11. Inventario de productos

Enunciado: Crea una clase `Inventario` que almacene objetos `Producto` y permita agregarlos y listarlos.

Explicación: Se usa una lista interna para manejar múltiples objetos.

Solución:

```
class Producto:
    def __init__(self, nombre, precio):
        self.nombre = nombre
        self.precio = precio

class Inventario:
    def __init__(self):
        self.productos = []
```

```
    def agregar_producto(self, producto):
        self.productos.append(producto)

    def listar_productos(self):
        for p in self.productos:
            print(f"{p.nombre}: ${p.precio}")

inv = Inventario()
inv.agregar_producto(Producto("Camisa", 25))
inv.agregar_producto(Producto("Pantalón", 40))
inv.listar_productos()
```

---

12. Clase Cuenta con historial

Enunciado: Amplía la clase `Cuenta` para guardar un historial de transacciones.

Explicación: Se guarda cada operación en una lista.

Solución:

```
class Cuenta:
    def __init__(self):
        self.saldo = 0
        self.historial = []

    def depositar(self, monto):
        self.saldo += monto
        self.historial.append(f"Depósito: {monto}")

    def retirar(self, monto):
        if monto <= self.saldo:
            self.saldo -= monto
            self.historial.append(f"Retiro: {monto}")
        else:
```

```
            self.historial.append("Intento de retiro fallido")

    def ver_historial(self):
        return self.historial

c = Cuenta()
c.depositar(100)
c.retirar(30)
print(c.ver_historial())
```

---

13. Clase Agenda

Enunciado: Crea una clase `Agenda` que permita agregar personas y buscar por nombre.

Explicación: Se utiliza una lista y un método de búsqueda.

Solución:

```
class Persona:
    def __init__(self, nombre, telefono):
        self.nombre = nombre
        self.telefono = telefono

class Agenda:
    def __init__(self):
        self.contactos = []

    def agregar(self, persona):
        self.contactos.append(persona)

    def buscar(self, nombre):
        for p in self.contactos:
            if p.nombre == nombre:
                return p.telefono
        return "No encontrado"
```

```
a = Agenda()
a.agregar(Persona("Laura", "1234"))
print(a.buscar("Laura"))  # 1234
```

---

14. Sistema de notas por asignatura

Enunciado: Implementa clases `Estudiante`, `Asignatura` y `Nota`.

Explicación: Se usan objetos anidados para representar relaciones.

Solución:

```
class Nota:
    def __init__(self, valor):
        self.valor = valor

class Asignatura:
    def __init__(self, nombre):
        self.nombre = nombre
        self.notas = []

    def agregar_nota(self, nota):
        self.notas.append(nota)

    def promedio(self):
        return sum(n.valor for n in self.notas) / len(self.notas)

class Estudiante:
    def __init__(self, nombre):
        self.nombre = nombre
        self.asignaturas = {}

    def agregar_asignatura(self, asignatura):
        self.asignaturas[asignatura.nombre] = asignatura
```

```
e = Estudiante("Juan")
mat = Asignatura("Matemáticas")
mat.agregar_nota(Nota(7))
mat.agregar_nota(Nota(9))
e.agregar_asignatura(mat)
print(e.asignaturas["Matemáticas"].promedio())  # 8.0
```

---

15. Contador compartido con clase

Enunciado: Crea una clase con un contador compartido entre todas las instancias.

Explicación: Usa una variable de clase.

Solución:

```
class ContadorCompartido:
    contador = 0

    def __init__(self):
        ContadorCompartido.contador += 1

    @classmethod
    def total(cls):
        return cls.contador

a = ContadorCompartido()
b = ContadorCompartido()
print(ContadorCompartido.total())  # 2
```

---

16. Sistema de reservas

Enunciado: Crea una clase `Reserva` que asigne asientos disponibles.

Explicación: Usa una lista de asientos ocupados y verifica disponibilidad.

Solución:

```
class Reserva:
```

```
    def __init__(self, total_asientos):
        self.total_asientos = total_asientos
        self.ocupados = []

    def reservar(self, asiento):
        if asiento in self.ocupados:
            return "Asiento ocupado"
        if asiento > self.total_asientos:
            return "Asiento no existe"
        self.ocupados.append(asiento)
        return "Reservado"

r = Reserva(10)
print(r.reservar(5))  # Reservado
print(r.reservar(5))  # Asiento ocupado
```

---

17. Clase Cronómetro

Enunciado: Crea una clase que simule un cronómetro con métodos `iniciar`, `parar`, `tiempo_transcurrido`.

Explicación: Usa la librería `time`.

Solución:

```
import time

class Cronometro:
    def iniciar(self):
        self.inicio = time.time()

    def parar(self):
        self.fin = time.time()

    def tiempo_transcurrido(self):
```

```
        return round(self.fin - self.inicio, 2)

c = Cronometro()
c.iniciar()
time.sleep(1.5)
c.parar()
print(c.tiempo_transcurrido())  # ~1.5
```

---

18. Simulación de dados

Enunciado: Crea una clase `Dado` que al tirar dé un número aleatorio entre 1 y 6.

Explicación: Se usa `random.randint`.

Solución:

```
import random

class Dado:
    def tirar(self):
        return random.randint(1, 6)

d = Dado()
print(d.tirar())  # número entre 1 y 6
```

---

19. Herencia con constructor

Enunciado: Crea clase `Animal` y una subclase `Perro` que use el constructor del padre y añada uno propio.

Explicación: Se usa `super()` para llamar al constructor base.

Solución:

```
class Animal:
    def __init__(self, especie):
        self.especie = especie

class Perro(Animal):
```

```
    def __init__(self, nombre):
        super().__init__("Perro")
        self.nombre = nombre

p = Perro("Fido")
print(p.especie, p.nombre)  # Perro Fido
```

---

20. Sistema de login

Enunciado: Implementa una clase `Usuario` con método `login` que valide usuario y contraseña.

Explicación: Se comparan credenciales simples.

Solución:

```
class Usuario:
    def __init__(self, usuario, contraseña):
        self.usuario = usuario
        self.contraseña = contraseña

    def login(self, usuario, contraseña):
        return self.usuario == usuario and self.contraseña == contraseña

u = Usuario("admin", "1234")
print(u.login("admin", "1234"))  # True
```

Dificultad avanzada

---

1. Sistema de Plugins Dinámicos

Enunciado: Crea una arquitectura de sistema que permita registrar y ejecutar plugins dinámicamente. Cada plugin debe implementar un método `run`.

Explicación: Utiliza clases base y herencia para permitir que múltiples clases plugin compartan una interfaz común.

Solución:

```
class PluginBase:
    def run(self):
        raise NotImplementedError

class PluginManager:
    def __init__(self):
        self.plugins = []

    def register(self, plugin):
        if isinstance(plugin, PluginBase):
            self.plugins.append(plugin)

    def run_all(self):
        for plugin in self.plugins:
            plugin.run()

class HelloPlugin(PluginBase):
    def run(self):
        print("Hello from plugin!")

manager = PluginManager()
manager.register(HelloPlugin())
manager.run_all()
```

---

2. Sistema Bancario con Transacciones

Enunciado: Diseña un sistema bancario con clases para `Cuenta`, `Transacción` y métodos para registrar depósitos, retiros y transferencias.

Explicación: Aplica encapsulamiento y abstracción para manejar transacciones.

Solución:

```
class Cuenta:
```

```
    def __init__(self, titular, saldo=0):
        self.titular = titular
        self._saldo = saldo

    def depositar(self, cantidad):
        self._saldo += cantidad

    def retirar(self, cantidad):
        if cantidad <= self._saldo:
            self._saldo -= cantidad
        else:
            raise ValueError("Saldo insuficiente")

    def transferir(self, otra_cuenta, cantidad):
        self.retirar(cantidad)
        otra_cuenta.depositar(cantidad)

    def mostrar_saldo(self):
        return self._saldo

cuenta1 = Cuenta("Ana", 1000)
cuenta2 = Cuenta("Luis", 500)
cuenta1.transferir(cuenta2, 300)
print(cuenta1.mostrar_saldo(), cuenta2.mostrar_saldo())
```

---

3. Gestor de Tareas con Prioridades

Enunciado: Implementa un sistema de gestión de tareas que permita establecer prioridades y ordenarlas automáticamente.

Explicación: Usa herencia y métodos mágicos (`__lt__`) para ordenar objetos.

Solución:

```
class Tarea:
```

```
    def __init__(self, descripcion, prioridad):
        self.descripcion = descripcion
        self.prioridad = prioridad

    def __lt__(self, otra):
        return self.prioridad < otra.prioridad

tareas = [
    Tarea("Estudiar", 2),
    Tarea("Dormir", 5),
    Tarea("Trabajar", 1)
]

tareas.sort()
for t in tareas:
    print(f"{t.descripcion} - Prioridad {t.prioridad}")
```

---

4. Editor de Texto con Deshacer/Rehacer

Enunciado: Crea un editor de texto que soporte operaciones `escribir`, `deshacer` y `rehacer`.

Explicación: Usa la pila (stack) para implementar el historial de comandos.

Solución:

```
class Editor:
    def __init__(self):
        self.texto = ""
        self.historial = []
        self.rehacer_stack = []

    def escribir(self, txt):
        self.historial.append(self.texto)
        self.texto += txt
        self.rehacer_stack.clear()
```

```
    def deshacer(self):
        if self.historial:
            self.rehacer_stack.append(self.texto)
            self.texto = self.historial.pop()

    def rehacer(self):
        if self.rehacer_stack:
            self.historial.append(self.texto)
            self.texto = self.rehacer_stack.pop()

editor = Editor()
editor.escribir("Hola")
editor.escribir(" Mundo")
editor.deshacer()
editor.rehacer()
print(editor.texto)
```

---

5. Modelado de Árbol Genealógico

Enunciado: Implementa una clase `Persona` que contenga padres e hijos y permita recorrer el árbol familiar.

Explicación: Usa estructuras recursivas y relaciones entre instancias.

Solución:

```
class Persona:
    def __init__(self, nombre):
        self.nombre = nombre
        self.hijos = []

    def agregar_hijo(self, hijo):
        self.hijos.append(hijo)
```

```
    def mostrar_arbol(self, nivel=0):
        print("  " * nivel + self.nombre)
        for hijo in self.hijos:
            hijo.mostrar_arbol(nivel + 1)

abuelo = Persona("Juan")
padre = Persona("Carlos")
hijo = Persona("Pedro")
abuelo.agregar_hijo(padre)
padre.agregar_hijo(hijo)
abuelo.mostrar_arbol()
```

---

6. Sistema de Archivos con Composite Pattern

Enunciado: Modela un sistema de archivos donde cada carpeta puede contener archivos o subcarpetas.

Explicación: Usa el patrón Composite para tratar archivos y carpetas de forma uniforme.

Solución:

```
class Componente:
    def mostrar(self, nivel=0):
        raise NotImplementedError

class Archivo(Componente):
    def __init__(self, nombre):
        self.nombre = nombre

    def mostrar(self, nivel=0):
        print("  " * nivel + self.nombre)

class Carpeta(Componente):
    def __init__(self, nombre):
        self.nombre = nombre
```

```
        self.contenido = []

    def agregar(self, componente):
        self.contenido.append(componente)

    def mostrar(self, nivel=0):
        print("  " * nivel + f"[{self.nombre}]")
        for c in self.contenido:
            c.mostrar(nivel + 1)

raiz = Carpeta("root")
raiz.agregar(Archivo("archivo1.txt"))
sub = Carpeta("docs")
sub.agregar(Archivo("doc1.pdf"))
raiz.agregar(sub)
raiz.mostrar()
```

---

7. Sistema de Ventas con Decorador

Enunciado: Aplica el patrón Decorador para añadir funcionalidades (como descuento o impuestos) a productos.

Explicación: Permite modificar comportamiento sin cambiar la clase base.

Solución:

```
class Producto:
    def __init__(self, nombre, precio):
        self.nombre = nombre
        self.precio = precio

    def calcular_precio(self):
        return self.precio

class ConImpuesto:
```

```
    def __init__(self, producto):
        self.producto = producto

    def calcular_precio(self):
        return self.producto.calcular_precio() * 1.21

producto = Producto("Libro", 100)
producto_con_iva = ConImpuesto(producto)
print(producto_con_iva.calcular_precio())
```

---

8. Simulador de Eventos con Observador

Enunciado: Implementa un sistema donde múltiples objetos reaccionan a un evento (p. ej., cambio de temperatura).

Explicación: Usa el patrón Observer para mantener sincronizados múltiples objetos.

Solución:

```
class Sensor:
    def __init__(self):
        self.observadores = []

    def registrar(self, obs):
        self.observadores.append(obs)

    def notificar(self, valor):
        for obs in self.observadores:
            obs.actualizar(valor)

class Termometro:
    def actualizar(self, temperatura):
        print(f"Temperatura actual: {temperatura}°C")

sensor = Sensor()
```

```
sensor.registrar(Termometro())
sensor.notificar(22)
```

---

9. Máquina de Estados Finita

Enunciado: Implementa una máquina de estados con transiciones entre "inicio", "en_proceso" y "terminado".

Explicación: Modela cada estado como un objeto diferente.

Solución:

```
class Estado:
    def siguiente(self, contexto):
        raise NotImplementedError

class Inicio(Estado):
    def siguiente(self, contexto):
        contexto.estado = EnProceso()

class EnProceso(Estado):
    def siguiente(self, contexto):
        contexto.estado = Terminado()

class Terminado(Estado):
    def siguiente(self, contexto):
        print("Proceso finalizado.")

class Proceso:
    def __init__(self):
        self.estado = Inicio()

    def avanzar(self):
        self.estado.siguiente(self)
```

```
p = Proceso()
p.avanzar()
p.avanzar()
p.avanzar()
```

---

10. Gestión de Recursos con Context Manager

Enunciado: Implementa una clase que maneje la conexión a un recurso usando `with`.

Explicación: Usa los métodos `__enter__` y `__exit__`.

Solución:

```
class Conexion:
    def __enter__(self):
        print("Conexión establecida")
        return self

    def __exit__(self, tipo, valor, traza):
        print("Conexión cerrada")

with Conexion() as c:
    print("Usando recurso")
```

---

11. Validador de Formularios

Enunciado: Implementa un validador que permita definir reglas personalizadas para los campos.

Explicación: Usa herencia para definir reglas y polimorfismo para validarlas.

Solución:

```
class Regla:
    def validar(self, valor):
        raise NotImplementedError

class NoVacio(Regla):
    def validar(self, valor):
        return bool(valor.strip())
```

```
class MinLongitud(Regla):
    def __init__(self, n):
        self.n = n

    def validar(self, valor):
        return len(valor) >= self.n

def validar_campo(valor, reglas):
    return all(regla.validar(valor) for regla in reglas)

print(validar_campo("hola", [NoVacio(), MinLongitud(3)]))
```

---

12. Sistema de Chat con Usuarios y Grupos

Enunciado: Modela un sistema de chat donde los usuarios pueden unirse a grupos y enviar mensajes.

Explicación: Usa relaciones entre clases y colecciones para manejar usuarios.

Solución:

```
class Usuario:
    def __init__(self, nombre):
        self.nombre = nombre

    def recibir(self, mensaje):
        print(f"{self.nombre} recibe: {mensaje}")

class Grupo:
    def __init__(self):
        self.usuarios = []

    def unir(self, usuario):
        self.usuarios.append(usuario)
```

```
    def enviar(self, mensaje):
        for u in self.usuarios:
            u.recibir(mensaje)

g = Grupo()
u1 = Usuario("Ana")
u2 = Usuario("Luis")
g.unir(u1)
g.unir(u2)
g.enviar("Hola a todos")
```

---

13. Contador de Palabras con Sobrecarga

Enunciado: Crea una clase que sobrecargue operadores para sumar y restar contadores de palabras.

Explicación: Implementa `__add__` y `__sub__`.

Solución:

```
class Contador:
    def __init__(self, palabras):
        self.palabras = palabras

    def __add__(self, otro):
        return Contador(self.palabras + otro.palabras)

    def __str__(self):
        return f"{len(self.palabras)} palabras"

c1 = Contador(["hola", "mundo"])
c2 = Contador(["python"])
c3 = c1 + c2
print(c3)
```

---

14. Control de Robots con Polimorfismo

Enunciado: Crea distintas clases de robots con distintos comportamientos de `mover`.

Explicación: Usa polimorfismo y clases base abstractas.

Solución:

```
class Robot:
    def mover(self):
        raise NotImplementedError

class RobotTerrestre(Robot):
    def mover(self):
        print("Moviéndose por tierra")

class RobotAcuatico(Robot):
    def mover(self):
        print("Nadando")

robots = [RobotTerrestre(), RobotAcuatico()]
for r in robots:
    r.mover()
```

---

15. Generador de Informes con Fábrica

Enunciado: Crea una fábrica que devuelva diferentes clases de informe según el formato solicitado.

Explicación: Usa el patrón Factory Method.

Solución:

```
class Informe:
    def generar(self):
        raise NotImplementedError

class InformePDF(Informe):
    def generar(self):
        return "Informe en PDF"
```

```
class InformeHTML(Informe):
    def generar(self):
        return "<h1>Informe HTML</h1>"

def crear_informe(tipo):
    if tipo == "pdf":
        return InformePDF()
    elif tipo == "html":
        return InformeHTML()

informe = crear_informe("html")
print(informe.generar())
```

---

16. Simulador de Procesos con Clases Abstractas

Enunciado: Crea una clase abstracta `Proceso` con métodos `iniciar()` y `terminar()`. Luego implementa varios tipos de procesos.

Explicación: Usa `abc.ABC` y `@abstractmethod` para definir interfaces obligatorias.

Solución:

```
from abc import ABC, abstractmethod

class Proceso(ABC):
    @abstractmethod
    def iniciar(self):
        pass

    @abstractmethod
    def terminar(self):
        pass
```

```
class ProcesoLento(Proceso):
    def iniciar(self):
        print("Inicio lento...")

    def terminar(self):
        print("Fin del proceso lento")

class ProcesoRapido(Proceso):
    def iniciar(self):
        print("Inicio rápido!")

    def terminar(self):
        print("Fin inmediato")

procesos = [ProcesoLento(), ProcesoRapido()]
for p in procesos:
    p.iniciar()
    p.terminar()
```

---

17. Simulación de Red Neuronal con Nodos

Enunciado: Crea una simulación de red neuronal simple donde cada `Nodo` recibe datos, los procesa y los transmite.

Explicación: Modela las conexiones como relaciones entre objetos con métodos de propagación.

Solución:

```
class Nodo:
    def __init__(self, nombre):
        self.nombre = nombre
        self.siguientes = []

    def conectar(self, otro):
        self.siguientes.append(otro)
```

```
    def procesar(self, entrada):
        salida = entrada * 2
        print(f"{self.nombre} procesó {entrada} -> {salida}")
        for nodo in self.siguientes:
            nodo.procesar(salida)

n1 = Nodo("Entrada")
n2 = Nodo("Oculto")
n3 = Nodo("Salida")

n1.conectar(n2)
n2.conectar(n3)

n1.procesar(1)
```

---

18. Gestor de Roles y Permisos

Enunciado: Implementa un sistema en el que los `Usuarios` tienen roles, y los roles contienen permisos.

Explicación: Usa composición y diccionarios para mapear roles a permisos.

Solución:

```
class Rol:
    def __init__(self, nombre, permisos):
        self.nombre = nombre
        self.permisos = permisos

class Usuario:
    def __init__(self, nombre, rol):
        self.nombre = nombre
        self.rol = rol
```

```
    def puede(self, permiso):
        return permiso in self.rol.permisos

admin = Rol("admin", {"leer", "escribir", "borrar"})
usuario = Usuario("Ana", admin)
print(usuario.puede("borrar"))  # True
```

---

19. Sistema de Publicación y Suscripción

Enunciado: Crea un sistema donde los `Publicadores` emiten eventos y los `Suscriptores` reaccionan a ellos.

Explicación: Simula un bus de eventos con suscripción dinámica.

Solución:

```
class Publicador:
    def __init__(self):
        self.suscriptores = []

    def suscribir(self, sub):
        self.suscriptores.append(sub)

    def emitir(self, evento):
        for sub in self.suscriptores:
            sub.recibir(evento)

class Suscriptor:
    def __init__(self, nombre):
        self.nombre = nombre

    def recibir(self, evento):
        print(f"{self.nombre} recibió: {evento}")

p = Publicador()
```

```
s1 = Suscriptor("Luis")
s2 = Suscriptor("Violeta")

p.suscribir(s1)
p.suscribir(s2)
p.emitir("¡Nuevo mensaje!")
```

---

20. Clonación de Objetos con Copia Profunda

Enunciado: Implementa un sistema de clonación de objetos complejos que puede contener listas u otros objetos.

Explicación: Usa el módulo `copy` para realizar copias profundas que no compartan referencias.

Solución:

```
import copy

class Documento:
    def __init__(self, titulo, paginas):
        self.titulo = titulo
        self.paginas = paginas

    def __str__(self):
        return f"{self.titulo}: {self.paginas}"

doc1 = Documento("Original", ["Página 1", "Página 2"])
doc2 = copy.deepcopy(doc1)
doc2.paginas.append("Página 3")

print(doc1)
print(doc2)
```

---

10. Tratamiento de ficheros
Dificultad facil

---

1. Crear un archivo de texto y escribir una línea

Enunciado:
Crea un archivo llamado `saludo.txt` y escribe la frase `"Hola, mundo"` en él.

Explicación:
Usamos `open()` con el modo de escritura `"w"` para crear el archivo y `write()` para escribir la frase.

Solución:

```
with open("saludo.txt", "w") as archivo:
    archivo.write("Hola, mundo")
```

---

2. Leer el contenido de un archivo

Enunciado:
Lee el contenido del archivo `saludo.txt` y muéstralo por pantalla.

Explicación:
Abrimos el archivo en modo lectura `"r"` y usamos `read()` para obtener su contenido.

Solución:

```
with open("saludo.txt", "r") as archivo:
    contenido = archivo.read()
    print(contenido)
```

---

3. Escribir varias líneas en un archivo

Enunciado:
Escribe tres líneas de texto distintas en un archivo llamado `frases.txt`.

Explicación:
Creamos una lista de líneas y usamos `writelines()` para escribirlas.

Solución:

```
lineas = ["Primera línea\n", "Segunda línea\n", "Tercera línea\n"]
with open("frases.txt", "w") as archivo:
    archivo.writelines(lineas)
```

---

4. Leer línea por línea

Enunciado:
Lee y muestra línea por línea el archivo `frases.txt`.

Explicación:
Usamos un bucle `for` para iterar por cada línea del archivo.

Solución:

```
with open("frases.txt", "r") as archivo:
    for linea in archivo:
        print(linea.strip())
```

---

5. Añadir contenido a un archivo existente

Enunciado:
Agrega la línea `"Cuarta línea"` al final del archivo `frases.txt`.

Explicación:
El modo `"a"` permite añadir texto sin borrar el contenido anterior.

Solución:

```
with open("frases.txt", "a") as archivo:
    archivo.write("Cuarta línea\n")
```

---

6. Contar las líneas de un archivo

Enunciado:
Cuenta cuántas líneas tiene el archivo `frases.txt`.

Explicación:
Leemos todas las líneas con `readlines()` y contamos cuántas hay.

Solución:

```
with open("frases.txt", "r") as archivo:
    lineas = archivo.readlines()
    print("Número de líneas:", len(lineas))
```

---

7. Copiar contenido de un archivo a otro

Enunciado:
Copia el contenido de `frases.txt` a un nuevo archivo `copia.txt`.

Explicación:
Leemos todo el contenido de un archivo y lo escribimos en otro.

Solución:

```
with open("frases.txt", "r") as original:
    contenido = original.read()
with open("copia.txt", "w") as copia:
    copia.write(contenido)
```

---

8. Comprobar si un archivo existe

Enunciado:
Verifica si el archivo `copia.txt` existe.

Explicación:
Usamos el módulo `os` para verificar la existencia del archivo.

Solución:

```
import os

if os.path.exists("copia.txt"):
    print("El archivo existe.")
else:
    print("El archivo no existe.")
```

---

9. Leer solo la primera línea

Enunciado:
Lee solo la primera línea del archivo `frases.txt`.

Explicación:
`readline()` devuelve solo la primera línea.

Solución:

```
with open("frases.txt", "r") as archivo:
    primera_linea = archivo.readline()
    print("Primera línea:", primera_linea.strip())
```

---

10. Leer archivo y convertir texto a mayúsculas

Enunciado:
Muestra el contenido de `frases.txt` en mayúsculas.

Explicación:
Se aplica `upper()` a cada línea leída.

Solución:

```
with open("frases.txt", "r") as archivo:
    for linea in archivo:
        print(linea.strip().upper())
```

---

11. Crear un archivo solo si no existe

Enunciado:
Crea el archivo `nuevo.txt` solo si no existe.

Explicación:
Usamos `os.path.exists` para verificar antes de crear.

Solución:

```
import os

if not os.path.exists("nuevo.txt"):
    with open("nuevo.txt", "w") as archivo:
        archivo.write("Este archivo es nuevo.")
```

---

12. Leer archivo completo y contar caracteres

Enunciado:
Cuenta el número total de caracteres en `frases.txt`.

Explicación:
Leemos el contenido completo y usamos `len()`.

Solución:

```
with open("frases.txt", "r") as archivo:
    contenido = archivo.read()
    print("Total de caracteres:", len(contenido))
```

---

13. Eliminar un archivo

Enunciado:
Elimina el archivo `copia.txt` si existe.

Explicación:
Usamos `os.remove()` para borrar un archivo.

Solución:

```
import os


if os.path.exists("copia.txt"):
    os.remove("copia.txt")
    print("Archivo eliminado.")
```

---

14. Escribir números del 1 al 10 en un archivo

Enunciado:
Escribe los números del 1 al 10, uno por línea, en `numeros.txt`.

Explicación:
Se usa un bucle para escribir cada número seguido de un salto de línea.

Solución:

```
with open("numeros.txt", "w") as archivo:
    for i in range(1, 11):
        archivo.write(f"{i}\n")
```

---

15. Leer un archivo y convertir a lista

Enunciado:
Convierte las líneas de `numeros.txt` en una lista de enteros.

Explicación:
Se leen las líneas, se les hace `strip()` y se convierten con `int()`.

Solución:

```
with open("numeros.txt", "r") as archivo:
    numeros = [int(linea.strip()) for linea in archivo]
    print(numeros)
```

---

16. Crear un archivo CSV básico

Enunciado:
Crea un archivo `datos.csv` con las columnas `nombre,edad` y dos filas de datos.

Explicación:
Un CSV es texto con columnas separadas por comas.

Solución:

```
with open("datos.csv", "w") as archivo:
    archivo.write("nombre,edad\n")
    archivo.write("Ana,30\n")
    archivo.write("Luis,25\n")
```

---

17. Leer un archivo CSV sin librerías

Enunciado:
Lee el archivo `datos.csv` e imprime cada fila como lista.

Explicación:
Se separan las líneas por coma con `split(",")`.

Solución:

```
with open("datos.csv", "r") as archivo:
    for linea in archivo:
        datos = linea.strip().split(",")
        print(datos)
```

---

18. Escribir fecha actual en un archivo

Enunciado:
Escribe la fecha actual en `fecha.txt`.

Explicación:
Usamos el módulo `datetime` para obtener la fecha de hoy.

Solución:

```
from datetime import date

hoy = date.today()
with open("fecha.txt", "w") as archivo:
```

```
    archivo.write(str(hoy))
```

---

19. Crear archivo con texto invertido

Enunciado:
Lee `saludo.txt` y guarda su contenido invertido en `invertido.txt`.

Explicación:
Se usa slicing `[::-1]` para invertir cadenas.

Solución:

```
with open("saludo.txt", "r") as entrada:
    texto = entrada.read()

with open("invertido.txt", "w") as salida:
    salida.write(texto[::-1])
```

---

20. Contar cuántas veces aparece una palabra

Enunciado:
Cuenta cuántas veces aparece la palabra `"línea"` en `frases.txt`.

Explicación:
Leemos el texto y usamos `count()`.

Solución:

```
with open("frases.txt", "r") as archivo:
    texto = archivo.read()
    print("Veces que aparece 'línea':", texto.count("línea"))
```

Dificultad media

---

1. Leer un archivo y contar líneas

Enunciado:
Lee un archivo de texto y cuenta cuántas líneas tiene.

Explicación:
Abrimos el archivo en modo lectura, usamos un bucle para contar las líneas y cerramos el archivo.

Solución:

```
def contar_lineas(nombre_archivo):
    with open(nombre_archivo, 'r') as f:
        lineas = f.readlines()
    return len(lineas)

# Ejemplo uso
print(contar_lineas('archivo.txt'))
```

---

2. Escribir una lista de cadenas en un archivo

Enunciado:
Dada una lista de cadenas, escríbelas en un archivo, cada una en una línea.

Explicación:
Abrimos el archivo en modo escritura, y usamos un bucle para escribir cada cadena seguida de un salto de línea.

Solución:

```
def escribir_lineas(nombre_archivo, lista):
    with open(nombre_archivo, 'w') as f:
        for linea in lista:
            f.write(linea + '\n')

# Ejemplo uso
escribir_lineas('salida.txt', ['Hola', 'Mundo', 'Python'])
```

---

3. Copiar contenido de un archivo a otro

Enunciado:
Copia todo el contenido de un archivo de texto a otro archivo.

Explicación:
Leemos todo el contenido del archivo origen y lo escribimos en el archivo destino.

Solución:

```
def copiar_archivo(origen, destino):
```

```
    with open(origen, 'r') as f1:
        contenido = f1.read()
    with open(destino, 'w') as f2:
        f2.write(contenido)

# Ejemplo uso
copiar_archivo('archivo1.txt', 'archivo2.txt')
```

---

4. Contar palabras en un archivo

Enunciado:
Cuenta cuántas palabras hay en un archivo de texto.

Explicación:
Leemos todo el archivo, lo dividimos por espacios para obtener palabras, y contamos.

Solución:

```
def contar_palabras(nombre_archivo):
    with open(nombre_archivo, 'r') as f:
        texto = f.read()
    palabras = texto.split()
    return len(palabras)

# Ejemplo uso
print(contar_palabras('archivo.txt'))
```

---

5. Buscar y reemplazar texto en un archivo

Enunciado:
Reemplaza todas las ocurrencias de una palabra en un archivo por otra palabra.

Explicación:
Leemos el contenido, hacemos el reemplazo con `.replace()`, y luego escribimos el contenido modificado.

Solución:

```
def reemplazar_texto(nombre_archivo, viejo, nuevo):
    with open(nombre_archivo, 'r') as f:
```

```
        contenido = f.read()
    contenido = contenido.replace(viejo, nuevo)
    with open(nombre_archivo, 'w') as f:
        f.write(contenido)

# Ejemplo uso
reemplazar_texto('archivo.txt', 'perro', 'gato')
```

---

6. Leer un archivo línea a línea sin cargar todo en memoria

Enunciado:
Lee un archivo línea a línea e imprime cada línea.

Explicación:
Abrimos el archivo y usamos un bucle for para iterar línea a línea sin cargar todo el archivo.

Solución:

```
def leer_linea_linea(nombre_archivo):
    with open(nombre_archivo, 'r') as f:
        for linea in f:
            print(linea.strip())

# Ejemplo uso
leer_linea_linea('archivo.txt')
```

---

7. Añadir texto al final de un archivo

Enunciado:
Añade una línea de texto al final de un archivo sin borrar su contenido.

Explicación:
Abrimos el archivo en modo append `'a'` para añadir contenido al final.

Solución:

```
def anadir_texto(nombre_archivo, texto):
    with open(nombre_archivo, 'a') as f:
        f.write(texto + '\n')
```

```
# Ejemplo uso
anadir_texto('archivo.txt', 'Esta es una línea añadida')
```

---

8. Leer un archivo CSV y mostrar contenido

Enunciado:
Lee un archivo CSV y muestra cada fila como lista.

Explicación:
Usamos el módulo `csv` para leer el archivo CSV fácilmente.

Solución:

```
import csv

def leer_csv(nombre_archivo):
    with open(nombre_archivo, newline='') as csvfile:
        lector = csv.reader(csvfile)
        for fila in lector:
            print(fila)

# Ejemplo uso
leer_csv('datos.csv')
```

---

9. Escribir datos en CSV desde una lista de listas

Enunciado:
Dada una lista de listas, escribe el contenido en un archivo CSV.

Explicación:
Usamos `csv.writer` para escribir filas en un archivo CSV.

Solución:

```
import csv

def escribir_csv(nombre_archivo, datos):
    with open(nombre_archivo, 'w', newline='') as csvfile:
        escritor = csv.writer(csvfile)
        escritor.writerows(datos)
```

```
# Ejemplo uso
datos = [['Nombre', 'Edad'], ['Ana', 23], ['Luis', 30]]
escribir_csv('salida.csv', datos)
```

---

10. Eliminar líneas vacías de un archivo

Enunciado:
Elimina todas las líneas vacías de un archivo.

Explicación:
Leemos las líneas, filtramos las no vacías, y escribimos de nuevo.

Solución:

```
def eliminar_lineas_vacias(nombre_archivo):
    with open(nombre_archivo, 'r') as f:
        lineas = f.readlines()
    lineas = [linea for linea in lineas if linea.strip() != '']
    with open(nombre_archivo, 'w') as f:
        f.writelines(lineas)

# Ejemplo uso
eliminar_lineas_vacias('archivo.txt')
```

---

11. Crear un archivo y escribir números del 1 al 100

Enunciado:
Crea un archivo y escribe los números del 1 al 100, uno por línea.

Explicación:
Usamos un bucle para escribir cada número con salto de línea.

Solución:

```
def escribir_numeros(nombre_archivo):
    with open(nombre_archivo, 'w') as f:
        for i in range(1, 101):
            f.write(str(i) + '\n')
```

```
# Ejemplo uso
escribir_numeros('numeros.txt')
```

---

12. Leer archivo binario y mostrar primeros 10 bytes

Enunciado:
Abre un archivo en modo binario y muestra los primeros 10 bytes.

Explicación:
Abrimos con `'rb'` y usamos `.read(10)`.

Solución:

```
def leer_binario(nombre_archivo):
    with open(nombre_archivo, 'rb') as f:
        bytes_10 = f.read(10)
    print(bytes_10)

# Ejemplo uso
leer_binario('imagen.jpg')
```

---

13. Contar caracteres de un archivo

Enunciado:
Cuenta el número total de caracteres (incluyendo espacios) en un archivo.

Explicación:
Leemos el archivo completo y usamos `len()`.

Solución:

```
def contar_caracteres(nombre_archivo):
    with open(nombre_archivo, 'r') as f:
        texto = f.read()
    return len(texto)

# Ejemplo uso
print(contar_caracteres('archivo.txt'))
```

---

14. Leer y mostrar un archivo en mayúsculas

Enunciado:
Lee un archivo y muestra su contenido todo en mayúsculas.

Explicación:
Leemos el archivo, convertimos a mayúsculas con `.upper()` y mostramos.

Solución:

```
def mostrar_mayusculas(nombre_archivo):
    with open(nombre_archivo, 'r') as f:
        texto = f.read()
    print(texto.upper())

# Ejemplo uso
mostrar_mayusculas('archivo.txt')
```

---

15. Extraer líneas que contienen una palabra

Enunciado:
Extrae y guarda en un nuevo archivo todas las líneas que contienen una palabra dada.

Explicación:
Leemos línea a línea y filtramos las que contienen la palabra.

Solución:

```
def extraer_lineas_palabra(origen, destino, palabra):
    with open(origen, 'r') as f:
        lineas = f.readlines()
    filtradas = [linea for linea in lineas if palabra in linea]
    with open(destino, 'w') as f:
        f.writelines(filtradas)

# Ejemplo uso
extraer_lineas_palabra('archivo.txt', 'filtradas.txt', 'Python')
```

---

16. Contar cuántas veces aparece una palabra en un archivo

Enunciado:
Cuenta el número total de apariciones de una palabra en un archivo.

Explicación:
Leemos el archivo, dividimos en palabras y contamos las ocurrencias.

Solución:

```
def contar_ocurrencias(nombre_archivo, palabra):
    with open(nombre_archivo, 'r') as f:
        texto = f.read().lower()
    palabras = texto.split()
    return palabras.count(palabra.lower())

# Ejemplo uso
print(contar_ocurrencias('archivo.txt', 'python'))
```

---

17. Leer archivo JSON y mostrar su contenido

Enunciado:
Lee un archivo JSON y muestra su contenido.

Explicación:
Usamos el módulo `json` para cargar el archivo y mostrarlo.

Solución:

```
import json

def leer_json(nombre_archivo):
    with open(nombre_archivo, 'r') as f:
        datos = json.load(f)
    print(datos)

# Ejemplo uso
leer_json('datos.json')
```

---

18. Guardar un diccionario en archivo JSON

Enunciado:
Guarda un diccionario Python en un archivo JSON.

Explicación:
Usamos `json.dump()` para escribir el diccionario en un archivo.

Solución:

```
import json


def guardar_json(nombre_archivo, diccionario):
    with open(nombre_archivo, 'w') as f:
        json.dump(diccionario, f, indent=4)


# Ejemplo uso
datos = {'nombre': 'Ana', 'edad': 28}
guardar_json('datos_guardados.json', datos)
```

---

19. Leer archivo y contar frecuencia de palabras

Enunciado:
Cuenta cuántas veces aparece cada palabra en un archivo y muestra un diccionario con las frecuencias.

Explicación:
Leemos el archivo, dividimos en palabras y usamos un diccionario para contar.

Solución:

```
def frecuencia_palabras(nombre_archivo):
    with open(nombre_archivo, 'r') as f:
        texto = f.read().lower()
    palabras = texto.split()
    freq = {}
    for p in palabras:
        freq[p] = freq.get(p, 0) + 1
    return freq
```

```
# Ejemplo uso
print(frecuencia_palabras('archivo.txt'))
```

---

20. Leer un archivo y mostrar las primeras N líneas

Enunciado:
Lee un archivo y muestra solo las primeras N líneas.

Explicación:
Usamos un contador para detenernos al llegar a N.

Solución:

```
def mostrar_primeras_lineas(nombre_archivo, n):
    with open(nombre_archivo, 'r') as f:
        for i, linea in enumerate(f):
            if i >= n:
                break
            print(linea.strip())

# Ejemplo uso
mostrar_primeras_lineas('archivo.txt', 5)
```

Dificultad avanzada

---

1. Leer un fichero CSV y convertirlo en una lista de diccionarios

Enunciado:
Lee un fichero llamado `empleados.csv` con el formato: `nombre,edad,salario`, y convierte cada fila en un diccionario, almacenándolos en una lista.

Explicación:
Utiliza `csv.DictReader` para convertir cada línea en un diccionario automáticamente.

Solución:

```
import csv
```

```
with open('empleados.csv', newline='', encoding='utf-8') as f:
    lector = csv.DictReader(f)
    empleados = list(lector)

print(empleados)
```

---

2. Contar las líneas no vacías de un fichero de texto

Enunciado:
Cuenta cuántas líneas **no vacías** hay en el fichero `poema.txt`.

Explicación:
Filtra las líneas ignorando aquellas que solo contienen espacios o están vacías.

Solución:

```
with open('poema.txt', 'r', encoding='utf-8') as f:
    lineas_no_vacias = [linea for linea in f if linea.strip()]

print(len(lineas_no_vacias))
```

---

3. Crear una copia cifrada de un fichero de texto (Cifrado César)

Enunciado:
Crea una copia de `mensaje.txt` cifrada con un desplazamiento de 3 posiciones.

Explicación:
El cifrado César cambia cada letra por otra desplazada en el alfabeto.

Solución:

```
def cifrado_cesar(texto, desplazamiento=3):
    resultado = ''
    for c in texto:
        if c.isalpha():
            base = ord('A') if c.isupper() else ord('a')
            resultado += chr((ord(c) - base + desplazamiento) % 26
+ base)
        else:
            resultado += c
```

```
    return resultado

with open('mensaje.txt', 'r', encoding='utf-8') as f_in, \
     open('mensaje_cifrado.txt', 'w', encoding='utf-8') as f_out:
    for linea in f_in:
        f_out.write(cifrado_cesar(linea))
```

---

4. Unir el contenido de varios ficheros `.txt` en uno solo

Enunciado:
Une todos los ficheros `.txt` de la carpeta actual en un solo fichero llamado `completo.txt`.

Explicación:
Recorre los ficheros con extensión `.txt` y concatena su contenido.

Solución:

```
import os

with open('completo.txt', 'w', encoding='utf-8') as f_out:
    for archivo in os.listdir():
        if archivo.endswith('.txt') and archivo != 'completo.txt':
            with open(archivo, 'r', encoding='utf-8') as f_in:
                f_out.write(f_in.read() + '\n')
```

---

5. Buscar la palabra más común en un fichero

Enunciado:
Encuentra la palabra más común en el fichero `libro.txt`.

Explicación:
Utiliza `collections.Counter` para contar palabras.

Solución:

```
from collections import Counter

with open('libro.txt', 'r', encoding='utf-8') as f:
    texto = f.read().lower()
```

```
palabras = [p.strip('.,;:()[]') for p in texto.split()]
contador = Counter(palabras)
mas_comun = contador.most_common(1)[0]
print(mas_comun)
```

---

6. Eliminar líneas duplicadas de un fichero

Enunciado:
Crea un nuevo fichero sin líneas duplicadas a partir de `datos.txt`.

Explicación:
Almacena líneas en un conjunto para eliminar duplicados.

Solución:

```
lineas_unicas = set()

with open('datos.txt', 'r', encoding='utf-8') as f:
    for linea in f:
        lineas_unicas.add(linea)

with open('datos_sin_duplicados.txt', 'w', encoding='utf-8') as f:
    f.writelines(lineas_unicas)
```

---

7. Dividir un fichero grande en partes de 100 líneas

Enunciado:
Divide el fichero `grande.txt` en partes de 100 líneas cada una.

Explicación:
Lee líneas por bloques y crea ficheros como `parte1.txt`, `parte2.txt`, etc.

Solución:

```
def dividir_fichero(nombre, lineas_por_parte):
    with open(nombre, 'r', encoding='utf-8') as f:
        contador = 1
        while True:
            lineas = list(f.readline() for _ in
range(lineas_por_parte))
```

```
            lineas = [l for l in lineas if l]
            if not lineas:
                break
            with open(f'parte{contador}.txt', 'w',
encoding='utf-8') as salida:
                salida.writelines(lineas)
            contador += 1

dividir_fichero('grande.txt', 100)
```

---

8. Leer un JSON y acceder a una clave anidada

Enunciado:
Lee el fichero `config.json` y muestra el valor de la clave `usuario -> nombre`.

Explicación:
Carga el JSON como diccionario y accede a la clave anidada.

Solución:

```
import json

with open('config.json', 'r', encoding='utf-8') as f:
    datos = json.load(f)

print(datos['usuario']['nombre'])
```

---

9. Detectar ficheros corruptos al leer archivos binarios

Enunciado:
Lee todos los ficheros `.dat` de una carpeta e imprime cuáles fallan al abrirse.

Explicación:
Controla errores de lectura con un `try/except`.

Solución:

```
import os

for archivo in os.listdir():
```

```
    if archivo.endswith('.dat'):
        try:
            with open(archivo, 'rb') as f:
                f.read()
        except Exception as e:
            print(f"Error al leer {archivo}: {e}")
```

---

10. Sustituir una palabra por otra en un fichero

Enunciado:
Sustituye la palabra "viejo" por "nuevo" en todo el fichero `reemplazar.txt`.

Explicación:
Lee el contenido, reemplaza, y guarda el resultado.

Solución:

```
with open('reemplazar.txt', 'r', encoding='utf-8') as f:
    contenido = f.read().replace('viejo', 'nuevo')

with open('reemplazar.txt', 'w', encoding='utf-8') as f:
    f.write(contenido)
```

---

11. Calcular el tamaño total de todos los ficheros `.log`

Enunciado:
Calcula el tamaño total (en bytes) de todos los archivos `.log` en la carpeta actual.

Explicación:
Usa `os.path.getsize()` para obtener el tamaño de cada fichero.

Solución:

```
import os

total = 0
for archivo in os.listdir():
    if archivo.endswith('.log'):
        total += os.path.getsize(archivo)
```

```
print(f"Tamaño total: {total} bytes")
```

---

12. Crear un índice de palabras con líneas donde aparecen

Enunciado:
Crea un índice de palabras del fichero `texto.txt`, indicando en qué líneas aparece cada una.

Explicación:
Guarda las palabras como claves y listas de números de línea como valores.

Solución:

```
from collections import defaultdict

indice = defaultdict(list)

with open('texto.txt', 'r', encoding='utf-8') as f:
    for num_linea, linea in enumerate(f, start=1):
        palabras = linea.strip().split()
        for palabra in set(palabras):
            indice[palabra].append(num_linea)

for palabra, lineas in indice.items():
    print(f"{palabra}: {lineas}")
```

---

13. Leer un fichero Excel y mostrar los valores de una columna

Enunciado:
Lee un archivo `datos.xlsx` y muestra los valores de la columna `A`.

Explicación:
Usa la librería `openpyxl` para leer hojas de cálculo de Excel.

Solución:

```
from openpyxl import load_workbook

wb = load_workbook('datos.xlsx')
hoja = wb.active
```

```
for fila in hoja.iter_rows(min_col=1, max_col=1,
values_only=True):
    print(fila[0])
```

---

14. Contar el número total de caracteres de un fichero grande sin cargarlo entero

Enunciado:
Cuenta el número total de caracteres del fichero `grande.txt` sin cargarlo completamente en memoria.

Explicación:
Lee por bloques para manejar ficheros grandes eficientemente.

Solución:

```
total_caracteres = 0

with open('grande.txt', 'r', encoding='utf-8') as f:
    while True:
        bloque = f.read(1024)
        if not bloque:
            break
        total_caracteres += len(bloque)

print(f"Total de caracteres: {total_caracteres}")
```

---

15. Validar que un fichero CSV tiene el mismo número de columnas en cada fila

Enunciado:
Verifica que todas las filas de `valores.csv` tienen el mismo número de columnas.

Explicación:
Compara el número de elementos de cada fila con el primero.

Solución:

```
valido = True
with open('valores.csv', 'r', encoding='utf-8') as f:
    filas = [linea.strip().split(',') for linea in f]
    columnas_esperadas = len(filas[0])
```

```
    for i, fila in enumerate(filas, 1):
        if len(fila) != columnas_esperadas:
            print(f"Error en línea {i}: {len(fila)} columnas")
            valido = False

if valido:
    print("El fichero es válido.")
```

---

16. Analizar un log y contar errores por tipo

Enunciado:
Analiza `log.txt` y cuenta cuántos errores de cada tipo (`ERROR`, `WARNING`, `INFO`) aparecen.

Explicación:
Busca patrones conocidos al inicio de cada línea.

Solución:

```
from collections import Counter

contador = Counter()

with open('log.txt', 'r', encoding='utf-8') as f:
    for linea in f:
        if linea.startswith(('ERROR', 'WARNING', 'INFO')):
            tipo = linea.split()[0]
            contador[tipo] += 1

print(contador)
```

---

17. Extraer y guardar imágenes incrustadas en un fichero binario

Enunciado:
Extrae todas las imágenes `.jpg` contenidas en un fichero binario `binario.dat` delimitadas por cabecera `FFD8` y pie `FFD9`.

Explicación:
Busca secuencias de bytes que coincidan con imágenes JPEG.

Solución:

```
with open('binario.dat', 'rb') as f:
    data = f.read()

inicio = 0
contador = 1
while True:
    inicio = data.find(b'\xff\xd8', inicio)
    if inicio == -1:
        break
    fin = data.find(b'\xff\xd9', inicio)
    if fin == -1:
        break
    fin += 2
    with open(f'imagen{contador}.jpg', 'wb') as img:
        img.write(data[inicio:fin])
    contador += 1
    inicio = fin
```

---

18. Guardar los nombres únicos de un fichero en orden alfabético

Enunciado:
Guarda en `nombres_unicos.txt` todos los nombres únicos del fichero `nombres.txt` en orden alfabético.

Explicación:
Usa un conjunto para eliminar duplicados y ordena la lista.

Solución:

```
with open('nombres.txt', 'r', encoding='utf-8') as f:
    nombres = set(linea.strip() for linea in f)

nombres_ordenados = sorted(nombres)

with open('nombres_unicos.txt', 'w', encoding='utf-8') as f:
```

```
    for nombre in nombres_ordenados:
        f.write(nombre + '\n')
```

---

19. Comprobar si dos ficheros binarios son idénticos

Enunciado:
Comprueba si `archivo1.bin` y `archivo2.bin` son exactamente iguales.

Explicación:
Lee ambos ficheros por bloques y compara sus contenidos.

Solución:

```
def son_iguales(f1, f2):
    with open(f1, 'rb') as a, open(f2, 'rb') as b:
        while True:
            bloque1 = a.read(1024)
            bloque2 = b.read(1024)
            if bloque1 != bloque2:
                return False
            if not bloque1:  # Ambos vacíos
                return True

print(son_iguales('archivo1.bin', 'archivo2.bin'))
```

---

20. Registrar actividad de un programa con marcas de tiempo

Enunciado:
Escribe en `log.txt` cada vez que se ejecute el script, con fecha y hora.

Explicación:
Usa `datetime.now()` para obtener la marca de tiempo.

Solución:

```
from datetime import datetime

with open('log.txt', 'a', encoding='utf-8') as f:
    f.write(f"Ejecutado en: {datetime.now()}\n")
```

Epílogo

Has llegado al final de este recorrido por el mundo de la programación en Python. Si estás leyendo estas líneas, es muy probable que hayas dedicado horas a resolver problemas, superar bloqueos y comprender conceptos que en un principio parecían complejos. Y eso, por sí solo, ya te convierte en alguien distinto a quien comenzó este libro: más hábil, más paciente, más capaz.

Este no es el fin, sino una puerta abierta. La programación es una herramienta poderosa que no solo resuelve problemas, sino que también transforma tu manera de pensar. Aprender a programar no es memorizar sintaxis, sino aprender a construir soluciones, a estructurar el pensamiento, a fallar y volver a intentar.

En estas páginas has trabajado con estructuras de control, funciones, objetos, ficheros, y más. Has entrenado tu lógica y tu creatividad. Pero sobre todo, has demostrado que puedes enfrentarte a la complejidad con perseverancia.

Te animo a seguir explorando. Crea tus propios proyectos, contribuye a código abierto, ayuda a quienes recién comienzan. Cada nueva línea de código puede ser una nueva oportunidad de aprendizaje. Porque en el fondo, programar no es solo escribir instrucciones: es una forma de dejar huella en el mundo digital.

Gracias por recorrer este camino. Que este libro sea solo el primer capítulo de una historia que tú decidirás escribir.

¡Sigue codificando!